

# **MAR GREGORIOS COLLEGE OF ARTS & SCIENCE**

Block No.8, College Road, Mogappair West, Chennai – 37

Affiliated to the University of Madras  
Approved by the Government of Tamil Nadu  
An ISO 9001:2015 Certified Institution



## **PG DEPARTMENT OF COMPUTER SCIENCE**

**SUBJECT NAME: ARTIFICIAL INTELLIGENCE**

**SUBJECT CODE: PSD3C**

**SEMESTER: III**

**PREPARED BY: PROF. D. SELVARAJ**

## Artificial Intelligence Syllabus

### UNIT I

Introduction - Intelligent Agents- Problem Solving - by Searching - Informed Search and Exploration - Constraint Satisfaction Problems - Adversarial Search

### UNIT II

Knowledge and Reasoning - Logical Agents - First-Order Logic - Inference in First-Order Logic - Knowledge Representation

### UNIT III

Planning – Planning and Acting in the Real World - Uncertain knowledge and reasoning - Uncertainty - Probabilistic Reasoning - Probabilistic Reasoning Over Time - Making Simple Decisions - Making Complex Decisions

### UNIT IV

Learning - Learning from Observations - Knowledge in Learning - Statistical Learning Methods - Reinforcement Learning

### UNIT V

Communicating, Perceiving, and Acting - Communication - Probabilistic Language Processing - Perception – Robotics.

### Text Books

(i) Stuart Russell and Peter Norvig, 2003, Artificial Intelligence: A Modern Approach, 2nd Edition, Prentice Hall of India, New Delhi.

### Reference Books

(i) Elaine Rich and Kevin Knight, 1991, Artificial Intelligence, 2nd Edition, Tata McGraw-Hill, New Delhi.

(ii) Herbert A. Simon, 1998, The Sciences of the Artificial Intelligence, 3rd Edition, MIT Press.

(iii) N.J. Nilson, 1983, Principles of AI, Springer Verlag.

### Website, E-learning resources

(i) <http://aima.eecs.berkeley.edu/slides-pdf/>

## ARTIFICIAL INTELLIGENCE

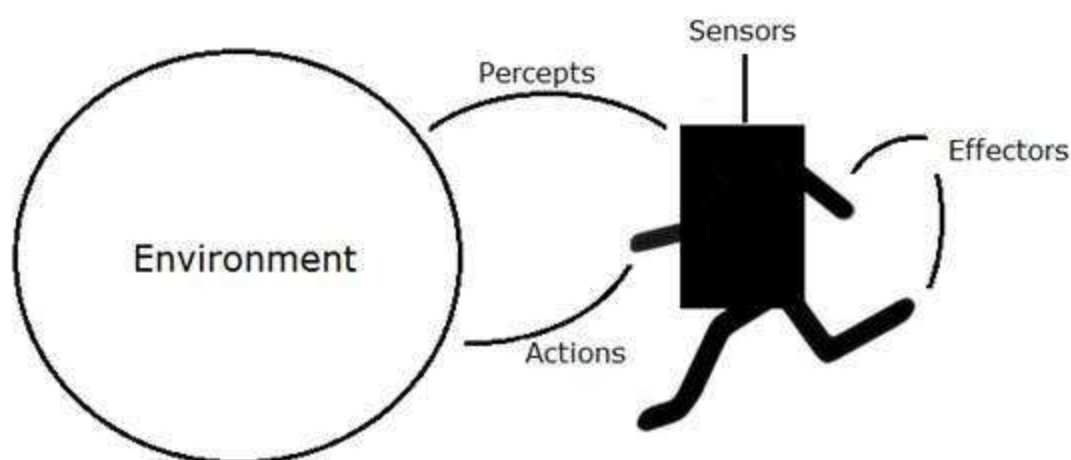
### UNIT I

An AI system is composed of an agent and its environment. The agents act in their environment. The environment may contain other agents.

#### **What are Agent and Environment?**

An **agent** is anything that can perceive its environment through **sensors** and acts upon that environment through **effectors**.

- A **human agent** has sensory organs such as eyes, ears, nose, tongue and skin parallel to the sensors, and other organs such as hands, legs, mouth, for effectors.
- A **robotic agent** replaces cameras and infrared range finders for the sensors, and various motors and actuators for effectors.
- A **software agent** has encoded bit strings as its programs and actions.



#### **Agent Terminology**

- **Performance Measure of Agent** – It is the criteria, which determines how successful an agent is.
- **Behavior of Agent** – It is the action that agent performs after any given sequence of percepts.
- **Percept** – It is agent's perceptual inputs at a given instance.

- **Percept Sequence** – It is the history of all that an agent has perceived till date.
- **Agent Function** – It is a map from the precept sequence to an action.

### Rationality

Rationality is nothing but status of being reasonable, sensible, and having good sense of judgment.

Rationality is concerned with expected actions and results depending upon what the agent has perceived. Performing actions with the aim of obtaining useful information is an important part of rationality.

### What is Ideal Rational Agent?

An ideal rational agent is the one, which is capable of doing expected actions to maximize its performance measure, on the basis of –

- Its percept sequence
- Its built-in knowledge base

Rationality of an agent depends on the following –

- The **performance measures**, which determine the degree of success.
- Agent's **Percept Sequence** till now.
- The agent's **prior knowledge about the environment**.
- The **actions** that the agent can carry out.

A rational agent always performs right action, where the right action means the action that causes the agent to be most successful in the given percept sequence. The problem the agent solves is characterized by Performance Measure, Environment, Actuators, and Sensors (PEAS).

### The Structure of Intelligent Agents

Agent's structure can be viewed as –

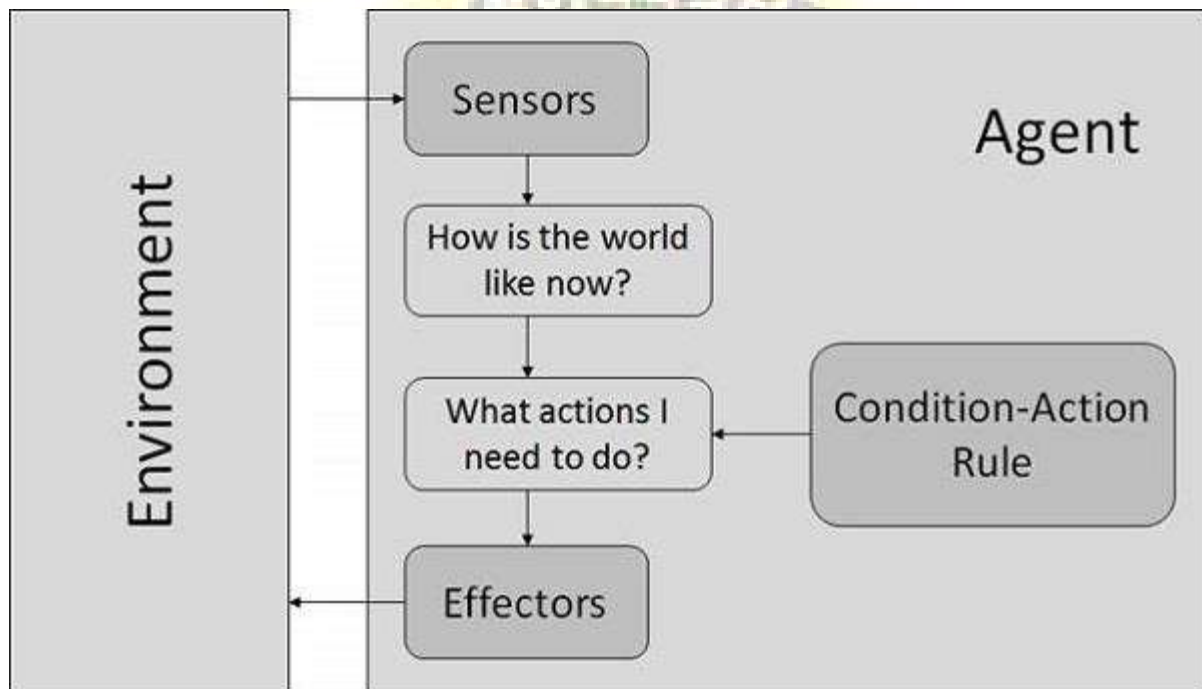
- Agent = Architecture + Agent Program
- Architecture = the machinery that an agent executes on.

- Agent Program = an implementation of an agent function.

### Simple Reflex Agents

- They choose actions only based on the current percept.
- They are rational only if a correct decision is made only on the basis of current percept.
- Their environment is completely observable.

**Condition-Action Rule** – It is a rule that maps a state (condition) to an action.



### Model Based Reflex Agents

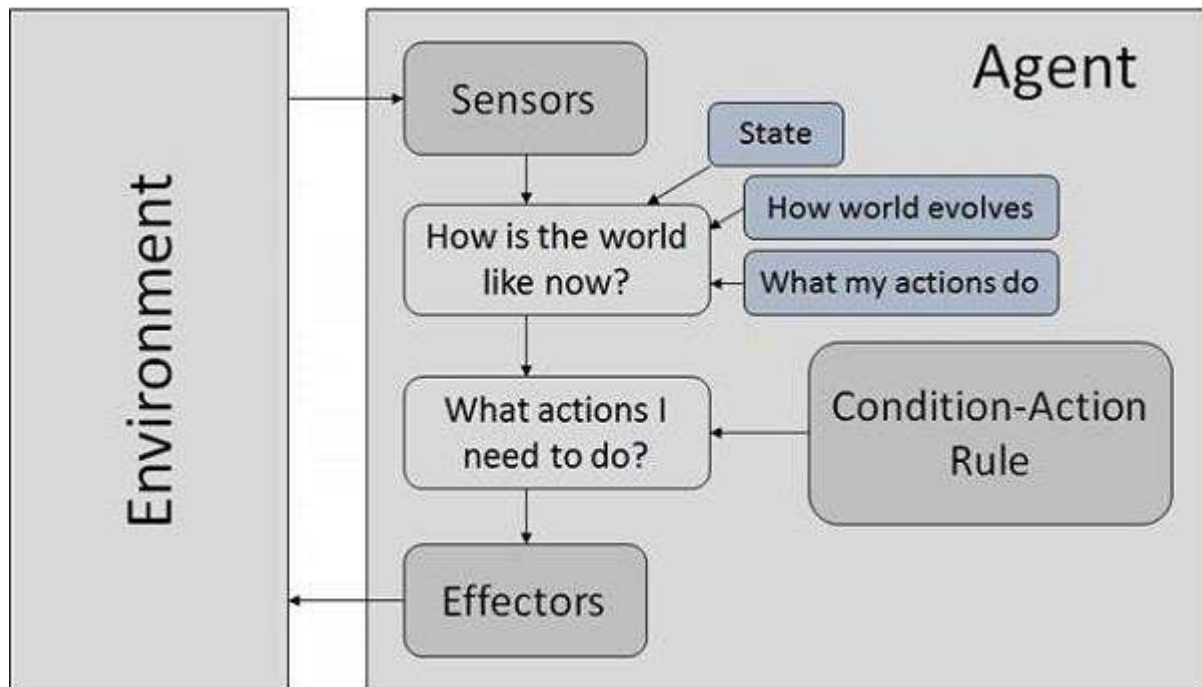
They use a model of the world to choose their actions. They maintain an internal state.

**Model** – knowledge about “how the things happen in the world”.

**Internal State** – It is a representation of unobserved aspects of current state depending on percept history.

**Updating the state requires the information about** –

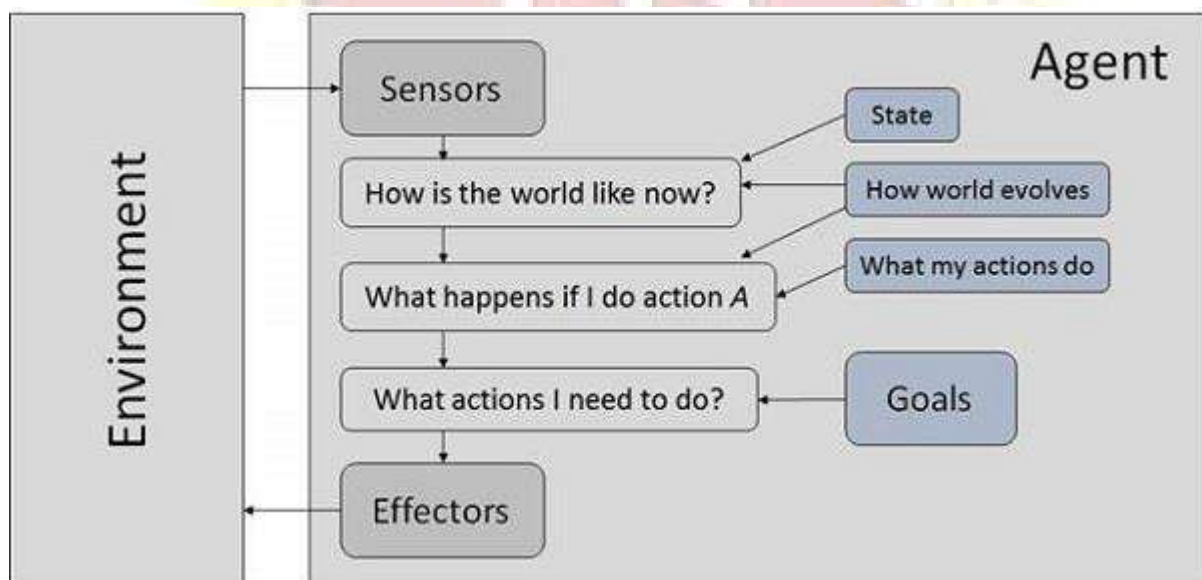
- How the world evolves.
- How the agent’s actions affect the world.



### Goal Based Agents

They choose their actions in order to achieve goals. Goal-based approach is more flexible than reflex agent since the knowledge supporting a decision is explicitly modeled, thereby allowing for modifications.

**Goal** – It is the description of desirable situations.

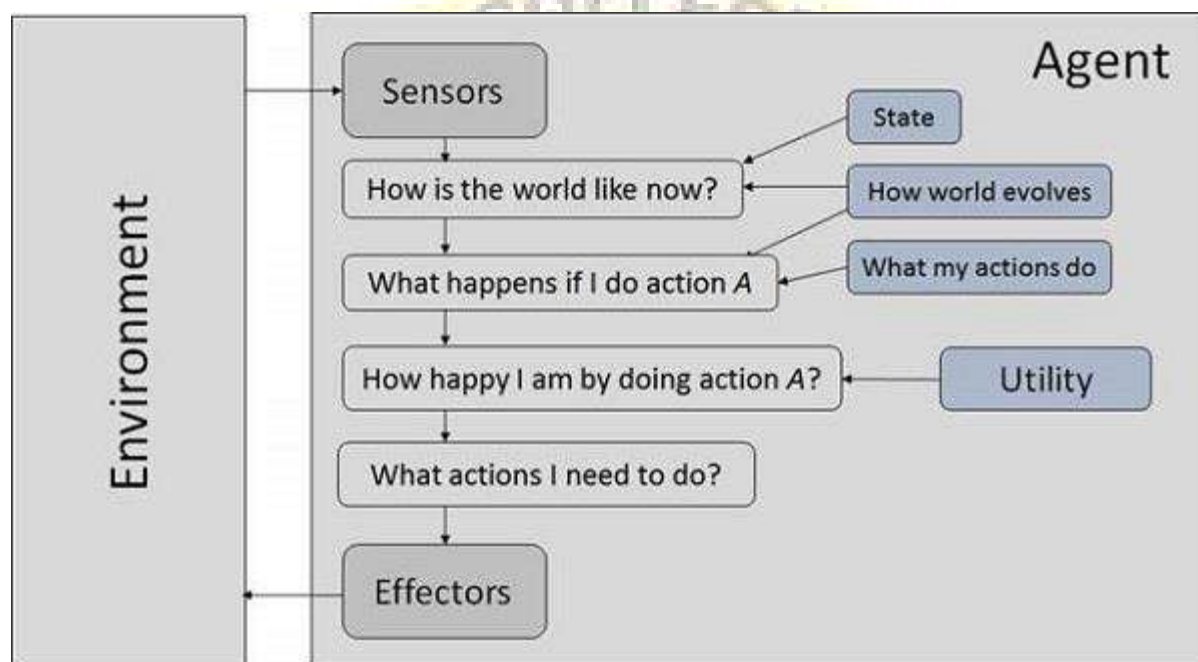


## Utility Based Agents

They choose actions based on a preference (utility) for each state.

Goals are inadequate when –

- There are conflicting goals, out of which only few can be achieved.
- Goals have some uncertainty of being achieved and you need to weigh likelihood of success against the importance of a goal.



## The Nature of Environments

Some programs operate in the entirely **artificial environment** confined to keyboard input, database, computer file systems and character output on a screen.

In contrast, some software agents (software robots or softbots) exist in rich, unlimited softbots domains. The simulator has a **very detailed, complex environment**. The software agent needs to choose from a long array of actions in real time. A softbot designed to scan the online preferences of the customer and show interesting items to the customer works in the **real** as well as an **artificial** environment.

The most famous **artificial environment** is the **Turing Test environment**, in which one real and other artificial agents are tested on equal ground. This is a very challenging environment as it is highly difficult for a software agent to perform as well as a human.

### Turing Test

The success of an intelligent behavior of a system can be measured with Turing Test.

Two persons and a machine to be evaluated participate in the test. Out of the two persons, one plays the role of the tester. Each of them sits in different rooms. The tester is unaware of who is machine and who is a human. He interrogates the questions by typing and sending them to both intelligences, to which he receives typed responses.

This test aims at fooling the tester. If the tester fails to determine machine's response from the human response, then the machine is said to be intelligent.

### Properties of Environment

The environment has multifold properties –

- **Discrete / Continuous** – If there are a limited number of distinct, clearly defined, states of the environment, the environment is discrete (For example, chess); otherwise it is continuous (For example, driving).
- **Observable / Partially Observable** – If it is possible to determine the complete state of the environment at each time point from the percepts it is observable; otherwise it is only partially observable.
- **Static / Dynamic** – If the environment does not change while an agent is acting, then it is static; otherwise it is dynamic.
- **Single agent / Multiple agents** – The environment may contain other agents which may be of the same or different kind as that of the agent.
- **Accessible / Inaccessible** – If the agent's sensory apparatus can have access to the complete state of the environment, then the environment is accessible to that agent.



- **Deterministic / Non-deterministic** – If the next state of the environment is completely determined by the current state and the actions of the agent, then the environment is deterministic; otherwise it is non-deterministic.
- **Episodic / Non-episodic** – In an episodic environment, each episode consists of the agent perceiving and then acting. The quality of its action depends just on the episode itself. Subsequent episodes do not depend on the actions in the previous episodes. Episodic environments are much simpler because the agent does not need to think ahead.

Searching is the universal technique of problem solving in AI. There are some single-player games such as tile games, Sudoku, crossword, etc. The search algorithms help you to search for a particular position in such games.

### Single Agent Pathfinding Problems

The games such as 3X3 eight-tile, 4X4 fifteen-tile, and 5X5 twenty four tile puzzles are single-agent-path-finding challenges. They consist of a matrix of tiles with a blank tile. The player is required to arrange the tiles by sliding a tile either vertically or horizontally into a blank space with the aim of accomplishing some objective.

The other examples of single agent pathfinding problems are Travelling Salesman Problem, Rubik's Cube, and Theorem Proving.

### Search Terminology

- **Problem Space** – It is the environment in which the search takes place. (A set of states and set of operators to change those states)
- **Problem Instance** – It is Initial state + Goal state.
- **Problem Space Graph** – It represents problem state. States are shown by nodes and operators are shown by edges.
- **Depth of a problem** – Length of a shortest path or shortest sequence of operators from Initial State to goal state.

- **Space Complexity** – The maximum number of nodes that are stored in memory.
- **Time Complexity** – The maximum number of nodes that are created.
- **Admissibility** – A property of an algorithm to always find an optimal solution.
- **Branching Factor** – The average number of child nodes in the problem space graph.
- **Depth** – Length of the shortest path from initial state to goal state.

### Brute-Force Search Strategies

They are most simple, as they do not need any domain-specific knowledge. They work fine with small number of possible states.

Requirements –

- State description
- A set of valid operators
- Initial state
- Goal state description

### Breadth-First Search

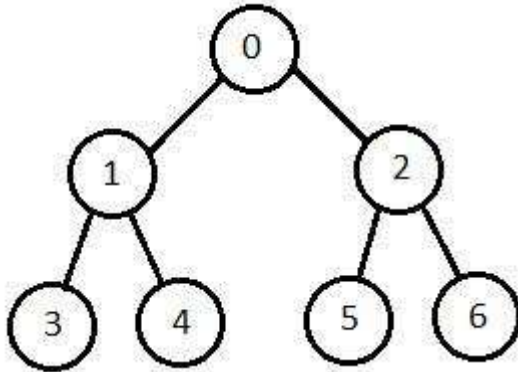
It starts from the root node, explores the neighboring nodes first and moves towards the next level neighbors. It generates one tree at a time until the solution is found. It can be implemented using FIFO queue data structure. This method provides shortest path to the solution.

If **branching factor** (average number of child nodes for a given node) =  $b$  and depth =  $d$ , then number of nodes at level  $d = b^d$ .

The total no of nodes created in worst case is  $b + b^2 + b^3 + \dots + b^d$ .

**Disadvantage** – Since each level of nodes is saved for creating next one, it consumes a lot of memory space. Space requirement to store nodes is exponential.

Its complexity depends on the number of nodes. It can check duplicate nodes.



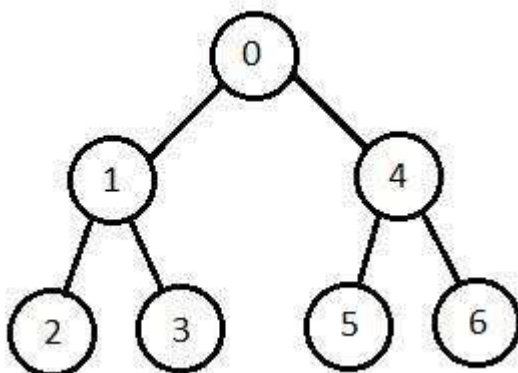
### Depth-First Search

It is implemented in recursion with LIFO stack data structure. It creates the same set of nodes as Breadth-First method, only in the different order.

As the nodes on the single path are stored in each iteration from root to leaf node, the space requirement to store nodes is linear. With branching factor  $b$  and depth as  $m$ , the storage space is  $bm$ .

**Disadvantage** – This algorithm may not terminate and go on infinitely on one path. The solution to this issue is to choose a cut-off depth. If the ideal cut-off is  $d$ , and if chosen cut-off is lesser than  $d$ , then this algorithm may fail. If chosen cut-off is more than  $d$ , then execution time increases.

Its complexity depends on the number of paths. It cannot check duplicate nodes.



### Bidirectional Search

It searches forward from initial state and backward from goal state till both meet to identify a common state.

The path from initial state is concatenated with the inverse path from the goal state. Each search is done only up to half of the total path.

### Uniform Cost Search

Sorting is done in increasing cost of the path to a node. It always expands the least cost node. It is identical to Breadth First search if each transition has the same cost.

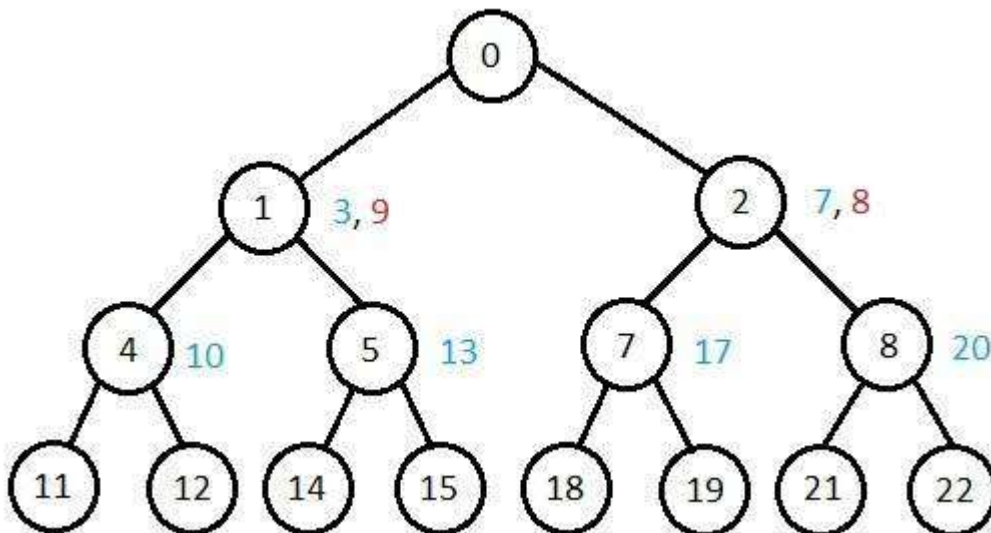
It explores paths in the increasing order of cost.

**Disadvantage** – There can be multiple long paths with the cost  $\leq C^*$ . Uniform Cost search must explore them all.

### Iterative Deepening Depth-First Search

It performs depth-first search to level 1, starts over, executes a complete depth-first search to level 2, and continues in such way till the solution is found.

It never creates a node until all lower nodes are generated. It only saves a stack of nodes. The algorithm ends when it finds a solution at depth  $d$ . The number of nodes created at depth  $d$  is  $b^d$  and at depth  $d-1$  is  $b^{d-1}$ .



## Comparison of Various Algorithms Complexities

Let us see the performance of algorithms based on various criteria –

Criterion	Breadth First	Depth First	Bidirectional	Uniform Cost	Interactive Deepening
Time	$b^d$	$b^m$	$b^{d/2}$	$b^d$	$b^d$
Space	$b^d$	$b^m$	$b^{d/2}$	$b^d$	$b^d$
Optimality	Yes	No	Yes	Yes	Yes
Completeness	Yes	No	Yes	Yes	Yes

### Informed (Heuristic) Search Strategies

To solve large problems with large number of possible states, problem-specific knowledge needs to be added to increase the efficiency of search algorithms.

### Heuristic Evaluation Functions

They calculate the cost of optimal path between two states. A heuristic function for sliding-tiles games is computed by counting number of moves that each tile makes from its goal state and adding these number of moves for all tiles.

### Pure Heuristic Search

It expands nodes in the order of their heuristic values. It creates two lists, a closed list for the already expanded nodes and an open list for the created but unexpanded nodes.

In each iteration, a node with a minimum heuristic value is expanded, all its child nodes are created and placed in the closed list. Then, the heuristic function is applied to the child nodes and they are placed in the open list according to their heuristic value. The shorter paths are saved and the longer ones are disposed.

## A \* Search

It is best-known form of Best First search. It avoids expanding paths that are already expensive, but expands most promising paths first.

$f(n) = g(n) + h(n)$ , where

- $g(n)$  the cost (so far) to reach the node
- $h(n)$  estimated cost to get from the node to the goal
- $f(n)$  estimated total cost of path through  $n$  to goal. It is implemented using priority queue by increasing  $f(n)$ .

## Greedy Best First Search

It expands the node that is estimated to be closest to goal. It expands nodes based on  $f(n) = h(n)$ . It is implemented using priority queue.

**Disadvantage** – It can get stuck in loops. It is not optimal.

## Local Search Algorithms

They start from a prospective solution and then move to a neighboring solution. They can return a valid solution even if it is interrupted at any time before they end.

## Hill-Climbing Search

It is an iterative algorithm that starts with an arbitrary solution to a problem and attempts to find a better solution by changing a single element of the solution incrementally. If the change produces a better solution, an incremental change is taken as a new solution. This process is repeated until there are no further improvements.

function Hill-Climbing (problem), returns a state that is a local maximum.

inputs: problem, a problem

local variables: current, a node

neighbor, a node

current  $\leftarrow$  Make\_Node(Initial-State[problem])

loop

do neighbor  $\leftarrow$  a highest\_valued successor of *current*

```

if Value[neighbor] ≤ Value[current] then
return State[current]
current <- neighbor

```

end

**Disadvantage** – This algorithm is neither complete, nor optimal.

### Local Beam Search

In this algorithm, it holds  $k$  number of states at any given time. At the start, these states are generated randomly. The successors of these  $k$  states are computed with the help of objective function. If any of these successors is the maximum value of the objective function, then the algorithm stops.

Otherwise the (initial  $k$  states and  $k$  number of successors of the states =  $2k$ ) states are placed in a pool. The pool is then sorted numerically. The highest  $k$  states are selected as new initial states. This process continues until a maximum value is reached.

function BeamSearch( *problem*,  $k$ ), returns a solution state.

start with  $k$  randomly generated states

loop

    generate all successors of all  $k$  states

    if any of the states = solution, then return the state

    else select the  $k$  best successors

end

### Simulated Annealing

Annealing is the process of heating and cooling a metal to change its internal structure for modifying its physical properties. When the metal cools, its new structure is seized, and the metal retains its newly obtained properties. In simulated annealing process, the temperature is kept variable.

We initially set the temperature high and then allow it to 'cool' slowly as the algorithm proceeds. When the temperature is high, the algorithm is allowed to accept worse solutions with high frequency.

Start

- Initialize  $k = 0$ ;  $L =$  integer number of variables;
- From  $i \rightarrow j$ , search the performance difference  $\Delta$ .
- If  $\Delta \leq 0$  then accept else if  $\exp(-\Delta/T(k)) > \text{random}(0,1)$  then accept;
- Repeat steps 1 and 2 for  $L(k)$  steps.
- $k = k + 1$ ;

Repeat steps 1 through 4 till the criteria is met.

End

### **Travelling Salesman Problem**

In this algorithm, the objective is to find a low-cost tour that starts from a city, visits all cities en-route exactly once and ends at the same starting city.

Start

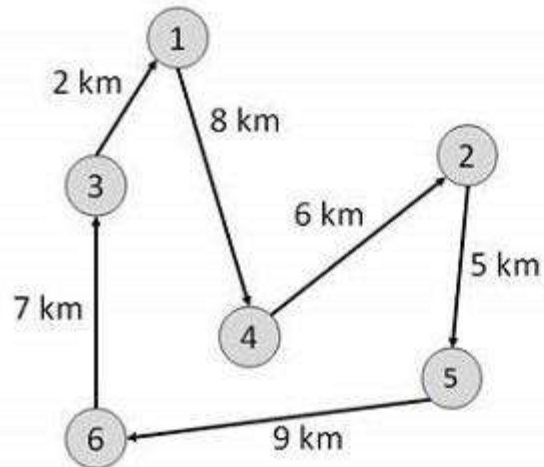
Find out all  $(n - 1)!$  Possible solutions, where  $n$  is the total number of cities.

Determine the minimum cost by finding out the cost of each of these  $(n - 1)!$  solutions.

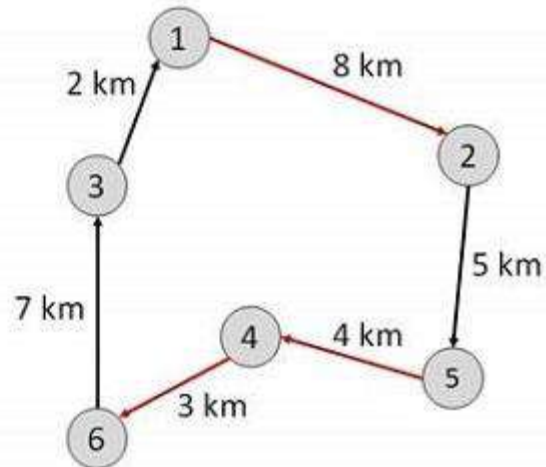
Finally, keep the one with the minimum cost.

end





Total Distance = 37km



Total Distance = 31km

Fuzzy Logic Systems (FLS) produce acceptable but definite output in response to incomplete, ambiguous, distorted, or inaccurate (fuzzy) input.

### What is Fuzzy Logic?

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

The conventional logic block that a computer can understand takes precise input and produces a definite output as TRUE or FALSE, which is equivalent to human's YES or NO.

The inventor of fuzzy logic, Lotfi Zadeh, observed that unlike computers, the human decision making includes a range of possibilities between YES and NO, such as –

CERTAINLY YES
POSSIBLY YES
CANNOT SAY
POSSIBLY NO
CERTAINLY NO

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

## Implementation

- It can be implemented in systems with various sizes and capabilities ranging from small micro-controllers to large, networked, workstation-based control systems.
- It can be implemented in hardware, software, or a combination of both.

## Why Fuzzy Logic?

Fuzzy logic is useful for commercial and practical purposes.

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

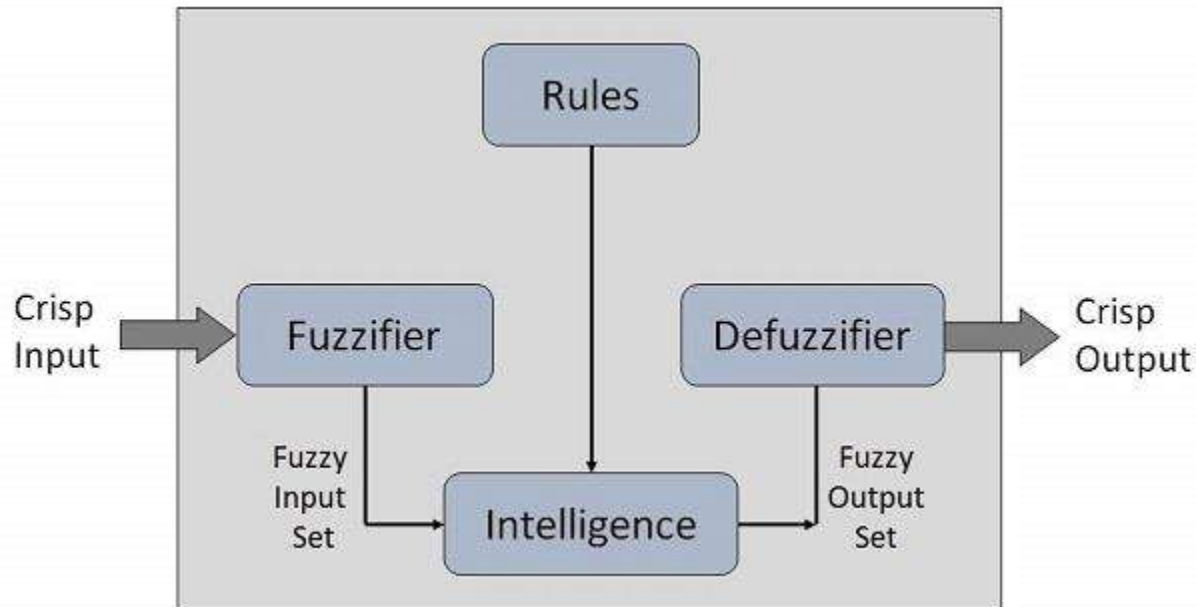
## Fuzzy Logic Systems Architecture

It has four main parts as shown –

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

<b>LP</b>	x	is	Large
			Positive
<b>MP</b>	x	is	Medium
			Positive
<b>S</b>	x	is	Small
<b>MN</b>	x	is	Medium
			Negative
<b>LN</b>	x	is	Large
			Negative

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



The **membership functions work on** fuzzy sets of variables.

### Membership Function

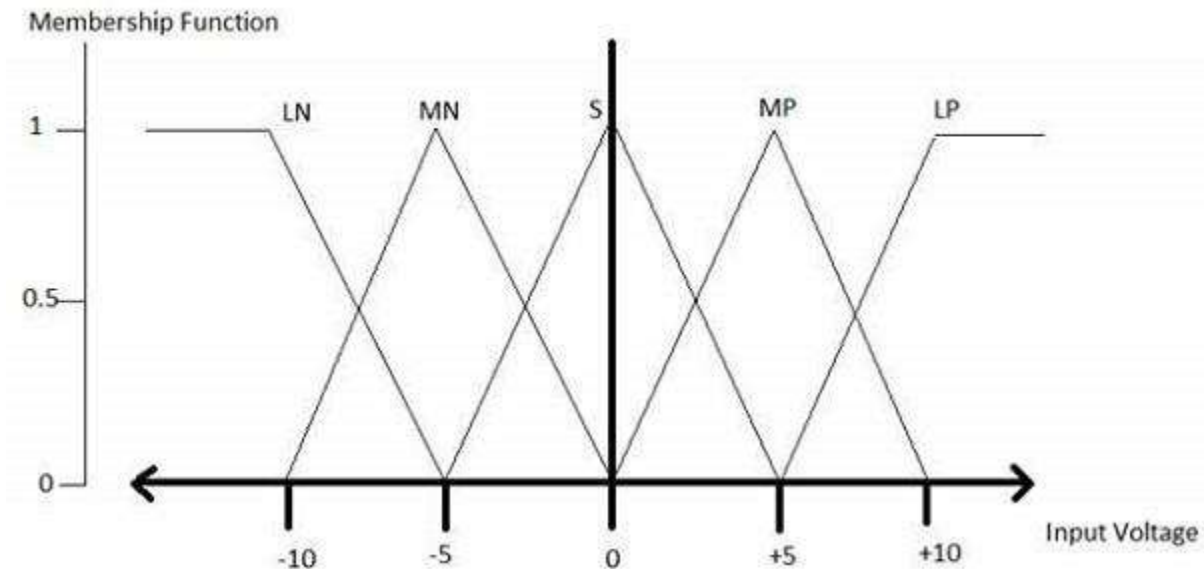
Membership functions allow you to quantify linguistic term and represent a fuzzy set graphically. A **membership function** for a fuzzy set  $A$  on the universe of discourse  $X$  is defined as  $\mu_A: X \rightarrow [0,1]$ .

Here, each element of  $X$  is mapped to a value between 0 and 1. It is called **membership value** or **degree of membership**. It quantifies the degree of membership of the element in  $X$  to the fuzzy set  $A$ .

- x axis represents the universe of discourse.
- y axis represents the degrees of membership in the  $[0, 1]$  interval.

There can be multiple membership functions applicable to fuzzify a numerical value. Simple membership functions are used as use of complex functions does not add more precision in the output.

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

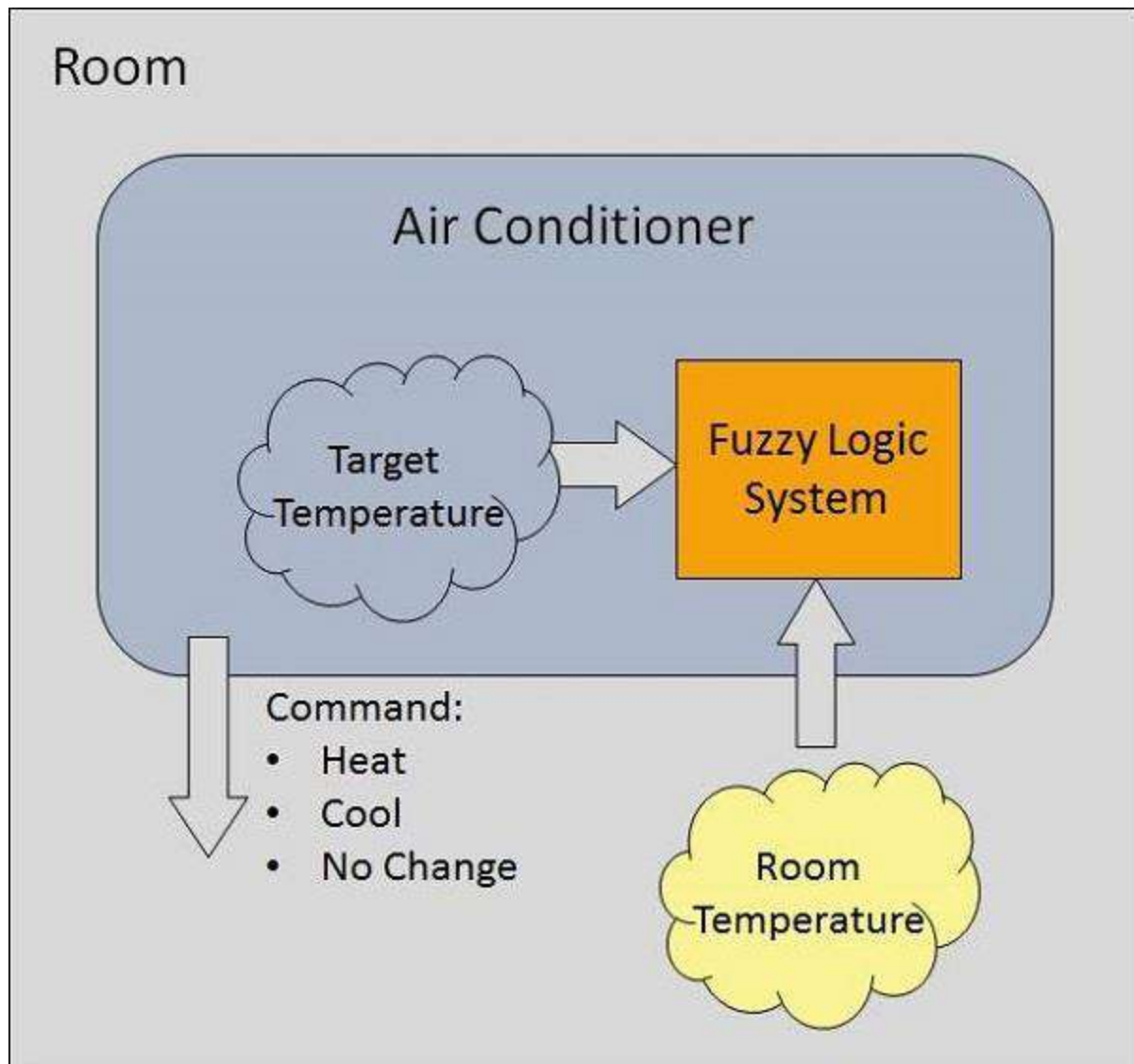


The triangular membership function shapes are most common among various other membership function shapes such as trapezoidal, singleton, and Gaussian.

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

### Example of a Fuzzy Logic System

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



### Algorithm

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

## Development

### Step 1 – Define linguistic variables and terms

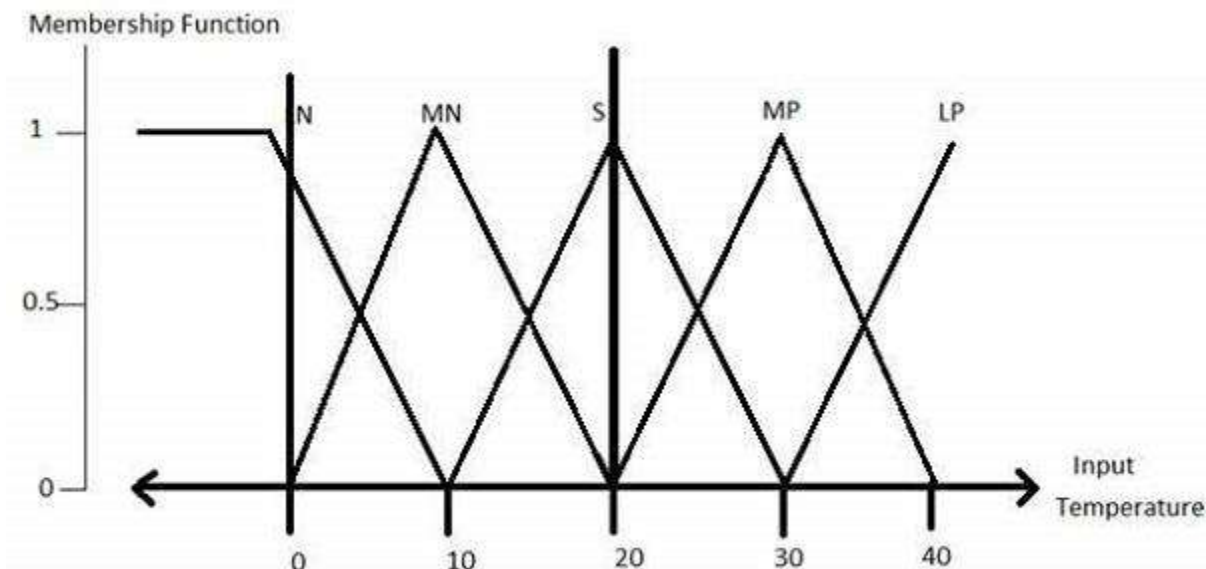
Linguistic variables are input and output variables in the form of simple words or sentences. For room temperature, cold, warm, hot, etc., are linguistic terms.

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

Every member of this set is a linguistic term and it can cover some portion of overall temperature values.

### Step 2 – Construct membership functions for them

The membership functions of temperature variable are as shown –



### Step3 – Construct knowledge base rules

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

RoomTemp. /Target	Very_Cold	Cold	Warm	Hot	Very_Hot

Very_Cold	No_Change	Heat	Heat	Heat	Heat
Cold	Cool	No_Change	Heat	Heat	Heat
Warm	Cool	Cool	No_Change	Heat	Heat
Hot	Cool	Cool	Cool	No_Change	Heat
Very_Hot	Cool	Cool	Cool	Cool	No_Change

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

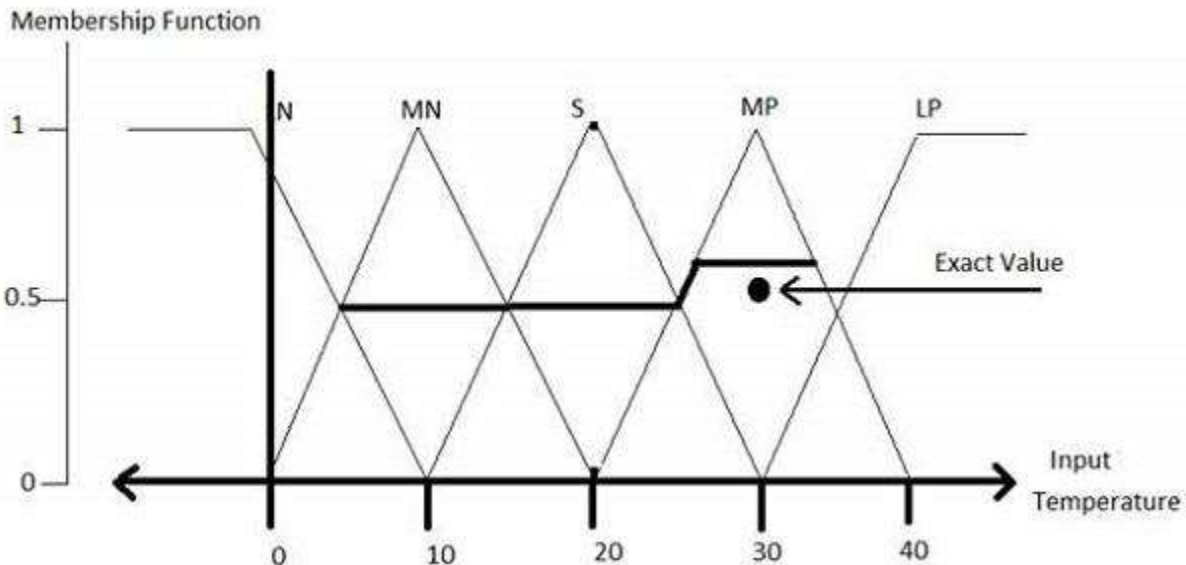
Sr. No.	Condition	Action
1	IF temperature=(Cold OR Very_Cold) AND target=Warm THEN	Heat
2	IF temperature=(Hot OR Very_Hot) AND target=Warm THEN	Cool
3	IF (temperature=Warm) AND (target=Warm) THEN	No_Change

**Step 4 – Obtain fuzzy value**

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

### Step 5 – Perform defuzzification

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



### Application Areas of Fuzzy Logic

The key application areas of fuzzy logic are as given –

#### Automotive Systems

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

#### Consumer Electronic Goods

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



### Domestic Goods

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

### Environment Control

- Air Conditioners/Dryers/Heaters
- Humidifiers

### Advantages of FLSs

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

### Disadvantages of FLSs

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

### Best First Search (Informed Search)

Prerequisites

: [BFS](#), [DFS](#)

In BFS and DFS, when we are at a node, we can consider any of the adjacent as next node. So both BFS and DFS blindly explore paths without considering any cost function. The idea of Best First Search is to use an evaluation function to decide which adjacent is most

promising and then explore. Best First Search falls under the category of Heuristic Search or Informed Search.

We use a priority queue to store costs of nodes. So the implementation is a variation of BFS, we just need to change Queue to PriorityQueue.

```
// This pseudocode is adapted from below
```

```
// source:
```

```
// https://courses.cs.washington.edu/
```

```
Best-First-Search(Graph g, Node start)
```

```
1) Create an empty PriorityQueue
```

```
    PriorityQueue pq;
```

```
2) Insert "start" in pq.
```

```
    pq.insert(start)
```

```
3) Until PriorityQueue is empty
```

```
    u = PriorityQueue.DeleteMin
```

```
    If u is the goal
```

```
        Exit
```

```
    Else
```

```
        Foreach neighbor v of u
```

```
            If v "Unvisited"
```

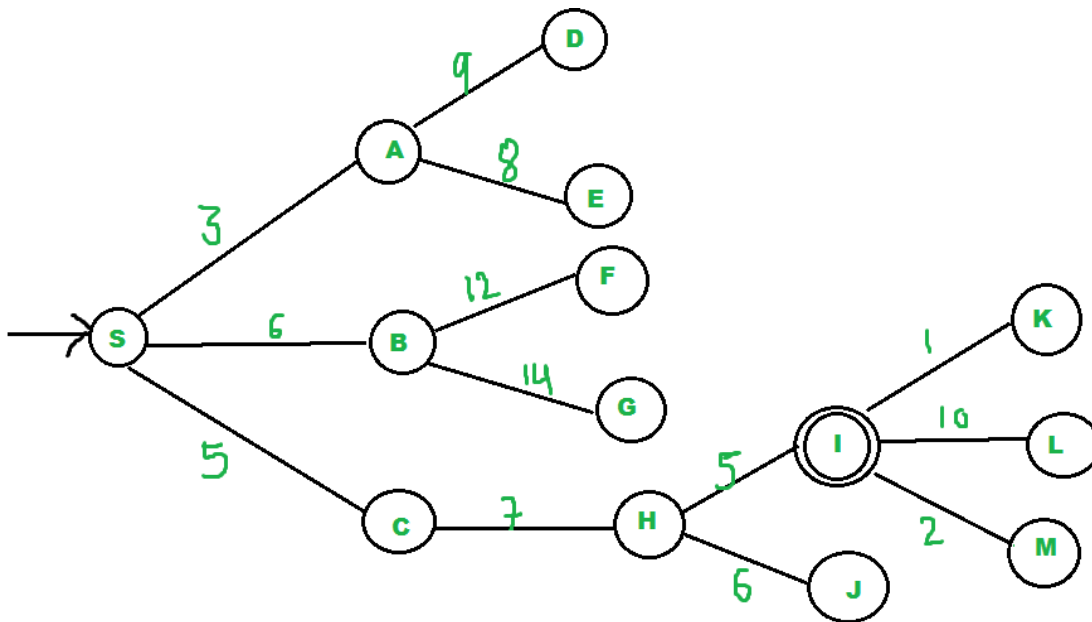
```
                Mark v "Visited"
```

```
                pq.insert(v)
```

```
            Mark u "Examined"
```

```
End procedure
```

Let us consider the below example.



We start from source "S" and search for goal "I" using given costs and Best First search.

pq initially contains S

We remove s from and process unvisited neighbors of S to pq.

pq now contains {A, C, B} (C is put before B because C has lesser cost)

We remove A from pq and process unvisited neighbors of A to pq.

pq now contains {C, B, E, D}

We remove C from pq and process unvisited neighbors of C to pq.

pq now contains {B, H, E, D}

We remove B from pq and process unvisited neighbors of B to pq.

pq now contains {H, E, D, F, G}

We remove H from pq. Since our goal "I" is a neighbor of H, we return.

### Constraint Satisfaction Problems in Artificial Intelligence

We have seen so many techniques like Local search, Adversarial search to solve different problems. The objective of every problem-solving technique is one, i.e., to find a solution to reach the goal. Although, in adversarial search and local search, there were no constraints on the agents while solving the problems and reaching to its solutions.

In this section, we will discuss another type of problem-solving technique known as Constraint satisfaction technique. By the name, it is understood that constraint satisfaction means *solving a problem under certain constraints or rules*.

*Constraint satisfaction is a technique where a problem is solved when its values satisfy certain constraints or rules of the problem.* Such type of technique leads to a deeper understanding of the problem structure as well as its complexity.

Constraint satisfaction depends on three components, namely:

- **X:** It is a set of variables.
- **D:** It is a set of domains where the variables reside. There is a specific domain for each variable.
- **C:** It is a set of constraints which are followed by the set of variables.

In constraint satisfaction, domains are the spaces where the variables reside, following the problem specific constraints. These are the three main elements of a constraint satisfaction technique. The constraint value consists of a pair of **{scope, rel}**. The **scope** is a tuple of variables which participate in the constraint and **rel** is a relation which includes a list of values which the variables can take to satisfy the constraints of the problem.

### Solving Constraint Satisfaction Problems

The requirements to solve a constraint satisfaction problem (CSP) is:

- A state-space
- The notion of the solution.

A state in state-space is defined by assigning values to some or all variables such as  $\{X_1=v_1, X_2=v_2, \text{ and so on...}\}$ .

**An assignment of values to a variable can be done 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, and the solution to the CSP remains consistent. Such assignment is known as Complete assignment.
- **Partial Assignment:** An assignment which assigns values to some of the variables only. Such type of assignments are called Partial assignments.

**Types of Domains in CSP**

**There are following two types of domains which are used by the variables :**

- **Discrete Domain:** It is an infinite domain which can have one state for multiple variables. **For example**, a start state can be allocated infinite times for each variable.
- **Finite Domain:** It is a finite domain which can have continuous states describing one domain for one specific variable. It is also called a continuous domain.

**Constraint Types in CSP**

With respect to the variables, basically there are following types of constraints:

- **Unary Constraints:** It is the simplest type of constraints that restricts the value of a single variable.
- **Binary Constraints:** It is the constraint type which relates two variables. A value  $x_2$  will contain a value which lies between  $x_1$  and  $x_3$ .
- **Global Constraints:** It is the constraint type which involves an arbitrary number of variables.

**Some special types of solution algorithms are used to solve the following types of constraints:**

- **Linear Constraints:** These type of constraints are commonly used in linear programming where each variable containing an integer value exists in linear form only.
- **Non-linear Constraints:** These type of constraints are used in non-linear programming where each variable (an integer value) exists in a non-linear form.

**Note:** A special constraint which works in real-world is known as **Preference constraint**.

**Constraint Propagation**

In local state-spaces, the choice is only one, i.e., to search for a solution. But in CSP, we have two choices either:

- We can search for a solution or
- We can perform a special type of inference called **constraint propagation**.

*Constraint propagation is a special type of inference which helps in reducing the legal number of values for the variables.* The idea behind constraint propagation is **local consistency**.

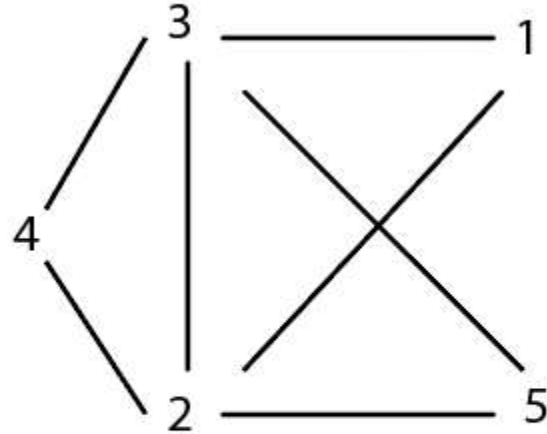
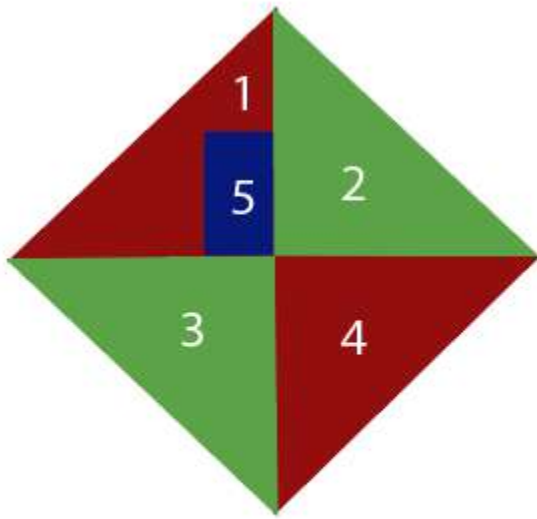
In local consistency, variables are treated as **nodes**, and each binary constraint is treated as an **arc** in the given problem. **There are following local consistencies which are discussed below:**

- **Node Consistency:** A single variable is said to be node consistent if all the values in the variable's domain satisfy the unary constraints on the variables.
- **Arc Consistency:** A variable is arc consistent if every value in its domain satisfies the binary constraints of the variables.
- **Path Consistency:** When the evaluation of a set of two variable with respect to a third variable can be extended over another variable, satisfying all the binary constraints. It is similar to arc consistency.
- **k-consistency:** This type of consistency is used to define the notion of stronger forms of propagation. Here, we examine the k-consistency of the variables.

### **CSP Problems**

Constraint satisfaction includes those problems which contains some constraints while solving the problem. CSP includes the following problems:

- **Graph Coloring:** The problem where the constraint is that no adjacent sides can have the same color.



## Graph Coloring

- **Sudoku Playing:** The gameplay where the constraint is that no number from 0-9 can be repeated in the same row or column.

## SUDOKU

4							5	9
2	6		5				3	
				9	2			
		2		6			1	
		3	8	1	9	7		
	7			3		5		
			3	4				
	3				6		2	7
5	9							6

Puzzle

4	1	7	6	8	3	2	5	9
2	6	9	5	7	1	8	3	4
3	8	5	4	9	2	6	7	1
8	4	2	7	6	5	9	1	3
6	5	3	8	1	9	7	4	2
9	7	1	2	3	4	5	6	8
7	2	6	3	4	8	1	9	5
1	3	8	9	5	6	4	2	7
5	9	4	1	2	7	3	8	6

Solution

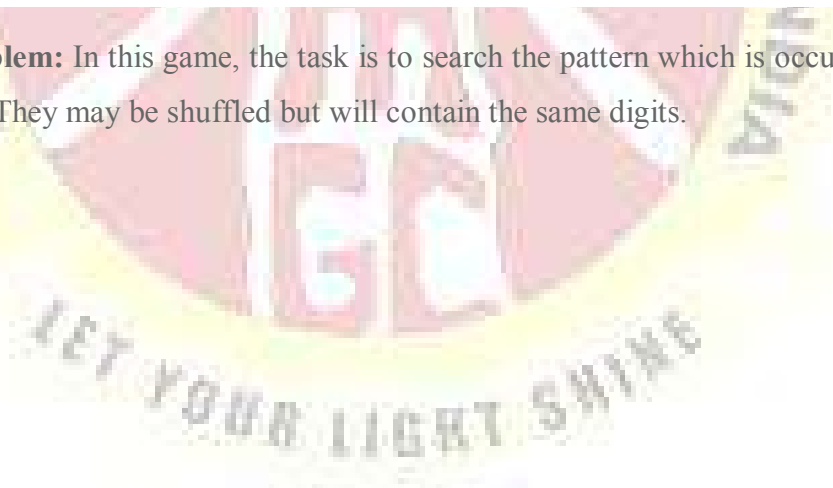
- **n-queen problem:** In n-queen problem, the constraint is that no queen should be placed either diagonally, in the same row or column.

Note: The n-queen problem is already discussed in Problem-solving in AI section.

- **Crossword:** In crossword problem, the constraint is that there should be the correct formation of the words, and it should be meaningful.



- **Latin square Problem:** In this game, the task is to search the pattern which is occurring several times in the game. They may be shuffled but will contain the same digits.





	1	1	1
1	2	3	4
1	3	4	2
1	4	2	3
1	2	4	3

	1	1	1	1
1	2	3	4	5
1	5	4	3	2
1	4	3	5	2
1	3	2	5	4

## Latin Sequence Problem

- **Cryptarithmic Problem:** This problem has one most important constraint that is, we cannot assign a different digit to the same character. All digits should contain a unique alphabet.

CONSTRAINT

SATISFACTION PROBLEMS

In which we see how treating states as more than just little black boxes leads to the invention of a range of powerful new search methods and a deeper understanding of problem structure and complexity.

Chapters 3 and 4 explored the idea that problems can be solved by searching in a space of states. These states can be evaluated by domain-specific heuristics and tested to see whether they are goal states. From the point of view of the search algorithm, however,

BLACK BOX each state is a black box with no discernible internal structure. It is represented by an arbitrary

data structure that can be accessed only by the problem-specific routines—the successor

function, heuristic function, and goal test.

This chapter examines constraint satisfaction problems, whose states and goal test

REPRESENTATION conform to a standard, structured, and very simple representation (Section 5.1). Search algorithms

can be defined that take advantage of the structure of states and use general-purpose

rather than problem-specific heuristics to enable the solution of large problems (Sections 5.2–

5.3). Perhaps most importantly, the standard representation of the goal test reveals the structure

of the problem itself (Section 5.4). This leads to methods for problem decomposition

and to an understanding of the intimate connection between the structure of a problem and

the difficulty of solving it.

## 5.1 CONSTRAINT SATISFACTION PROBLEMS

Formally speaking, a constraint satisfaction problem (or CSP) is defined by a set of variables and constraints.

SATISFACTION

## PROBLEM

VARIABLES  $x_1, x_2, \dots, x_n$ , and a set of constraints,  $C_1, C_2, \dots, C_m$ . Each variable  $x_i$  has a

CONSTRAINTS nonempty domain  $D_i$  of possible values. Each constraint  $C_i$  involves some subset of the

## DOMAIN

## VALUES

variables and specifies the allowable combinations of values for that subset. A state of the problem is defined by an assignment of values to some or all of the variables,  $\{x_i = v_i, x_j =$

ASSIGNMENT  $v_j, \dots\}$ . An assignment that does not violate any constraints is called a consistent or legal

CONSISTENT assignment. A complete assignment is one in which every variable is mentioned, and a so-

lution to a CSP is a complete assignment that satisfies all the constraints. Some CSPs also require a solution that maximizes an objective function.

o what does all this mean? Suppose that, having tired of Romania, we are looking

at a map of Australia showing each of its states and territories, as in Figure 5.1(a), and that

we are given the task of coloring each region either red, green, or blue in such a way that no

neighboring regions have the same color. To formulate this as a CSP, we define the variables

to be the regions: WA, NT, Q, NSW, V, SA, and T. The domain of each variable is the set

{red, green, blue}. The constraints require neighboring regions to have distinct colors; for

example, the allowable combinations for WA and NT are the pairs

{(red, green), (red, blue), (green, red), (green, blue), (blue, red), (blue, green)} .

(The constraint can also be represented more succinctly as the inequality  $WA \neq NT$ , pro-

vided the constraint satisfaction algorithm has some way to evaluate such expressions.)

There

are many possible solutions, such as

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

**CONSTRAINT GRAPH** It is helpful to visualize a CSP as a constraint graph, as shown in Figure 5.1(b). The nodes

of the graph correspond to variables of the problem and the arcs correspond to constraints.

Treating a problem as a CSP confers several important benefits. Because the representa-

tion of states in a CSP conforms to a standard pattern—that is, a set of variables with assigned

values—the successor function and goal test can be written in a generic way that applies to all

CSPs. Furthermore, we can develop effective, generic heuristics that require no additional,

domain-specific expertise. Finally, the structure of the constraint graph can be used to sim-

plify the solution process, in some cases giving an exponential reduction in complexity. The

viewed as a constraint satisfaction problem. The goal is to assign colors to each region so

that no neighboring regions have the same color. (b) The map-coloring problem represented

as a constraint graph.

**It is fairly easy to see that a CSP can be given an incremental formulation as a standard search problem as follows:**

- ◆ **Initial state:** the empty assignment {}, in which all variables are unassigned.
- ◆ **Successor function:** a value can be assigned to any unassigned variable, provided that it does not conflict with previously assigned variables.
- ◆ **Goal test:** the current assignment is complete.
- ◆ **Path cost:** a constant cost (e.g., 1) for every step.

**Every solution must be a complete assignment and therefore appears at depth  $n$  if there are  $n$  variables. Furthermore, the search tree extends only to depth  $n$ . For these reasons, depth-first search algorithms are popular for CSPs. (See Section 5.2.) It is also the case that the path by which a solution is reached is irrelevant. Hence, we can also use a complete-state formulation, in which every state is a complete assignment that might or might not satisfy the constraints. Local search methods work well for this formulation. (See Section 5.3.)**

**FINITE DOMAINS** The simplest kind of CSP involves variables that are discrete and have finite domains.

Map-coloring problems are of this kind. The 8-queens problem described in Chapter 3 can also be viewed as a finite-domain CSP, where the variables  $Q_1, \dots, Q_8$  are the positions of each queen in columns  $1, \dots, 8$  and each variable has the domain  $\{1, 2, 3, 4, 5, 6, 7, 8\}$ . If the

maximum domain size of any variable in a CSP is  $d$ , then the number of possible complete assignments is  $O(d^n)$

$n$

)—that is, exponential in the number of variables. Finite-domain CSPs

**BOOLEAN CSPS** include Boolean CSPs, whose variables can be either true or false. Boolean CSPs include

as special cases some NP-complete problems, such as 3SAT. (See Chapter 7.) In the worst case, therefore, we cannot expect to solve finite-domain CSPs in less than exponential time.

In most practical applications, however, general-purpose CSP algorithms can solve problems

orders of magnitude larger than those solvable via the general-purpose search algorithms that

we saw in Chapter 3.

**INFINITE DOMAINS** Discrete variables can also have infinite domains—for example, the set of integers or

the set of strings. For example, when scheduling construction jobs onto a calendar, each job's

start date is a variable and the possible values are integer numbers of days from the current

date. With infinite domains, it is no longer possible to describe constraints by enumerating all allowed combinations of values. Instead, a constraint language must be used. For ex-

**CONSTRAINT**

**LANGUAGE**

ample, if Job1, which takes five days, must precede Job3, then we would need a constraint

language of algebraic inequalities such as  $\text{StartJob1} + 5 \leq \text{StartJob3}$ . It is also no longer

possible to solve such constraints by enumerating all possible assignments, because there are

infinitely many of them. Special solution algorithms (which we will not discuss here) exist

for linear constraints on integer variables—that is, constraints, such as the one just given,

**LINEAR**

**CONSTRAINTS**

in which each variable appears only in linear form. It can be shown that no algorithm exists

for solving general nonlinear constraints on integer variables. In some cases, we can reduce

**NONLINEAR**

**CONSTRAINTS**

integer constraint problems to finite-domain problems simply by bounding the values of all

the variables. For example, in a scheduling problem, we can set an upper bound equal to the

total length of all the jobs to be scheduled.

Constraint satisfaction problems with continuous domains are very common in the real

**CONTINUOUS**

**DOMAINS**

world and are widely studied in the field of operations research. For example, the scheduling

of experiments on the Hubble Space Telescope requires very precise timing of observations;

the start and finish of each observation and maneuver are continuous-valued variables that must obey a variety of astronomical, precedence, and power constraints. The best-known

category of continuous-domain CSPs is that of linear programming problems, where con-

### LINEAR PROGRAMMING

straints must be linear inequalities forming a convex region. Linear programming problems

can be solved in time polynomial in the number of variables. Problems with different types of

constraints and objective functions have also been studied—quadratic programming, second-

order conic programming, and so on.

In addition to examining the types of variables that can appear in CSPs, it is useful to

look at the types of constraints. The simplest type is the unary constraint, which restricts the

value of a single variable. For example, it could be the case that South Australians actively

dislike the color green. Every unary constraint can be eliminated simply by preprocessing

the domain of the corresponding variable to remove any value that violates the constraint.

A

**BINARY CONSTRAINT** binary constraint relates two variables. For example, SA ≠ NSW is a binary constraint. A



binary CSP is one with only binary constraints; it can be represented as a constraint graph, as

in Figure 5.1(b).

Higher-order constraints involve three or more variables. A familiar example is provided by cryptarithmic puzzles. (See Figure 5.2(a).) It is usual to insist that each letter in

a cryptarithmic puzzle represent a different digit. For the case in Figure 5.2(a)), this would

be represented as the six-variable constraint Alldiff (F, T, U, W, R, O). Alternatively, it can be represented by a collection of binary constraints such as  $F \neq T$ . The addition constraints

on the four columns of the puzzle also involve several variables and can be written as

$$O + O = R + 10 \cdot X_1$$

$$X_1 + W + W = U + 10 \cdot X_2$$

$$X_2 + T + T = O + 10 \cdot X_3$$

$$X_3 = F$$

where  $X_1$ ,  $X_2$ , and  $X_3$  are auxiliary variables representing the digit (0 or 1) carried over into AUXILIARY

VARIABLES

the next column. Higher-order constraints can be represented in a constraint hypergraph,

CONSTRAINT

HYPERGRAPH

such as the one shown in Figure 5.2(b). The sharp-eyed reader will have noticed that the

Alldiff constraint can be broken down into binary constraints— $F \neq T$ ,  $F \neq U$ , and so on.

In fact, as Exercise 5.11 asks you to prove, every higher-order, finite-domain constraint can be reduced to a set of binary constraints if enough auxiliary variables are introduced. Because

of this, we will deal only with binary constraints in this chapter.

The constraints we have described so far have all been absolute constraints, violation PREFERENCE of which rules out a potential solution. Many real-world CSPs include preference constraints

indicating which solutions are preferred. For example, in a university timetabling problem, Prof. X might prefer teaching in the morning whereas Prof. Y prefers teaching in the afternoon. A timetable that has Prof. X teaching at 2 p.m. would still be a solution (unless Prof. X

happens to be the department chair), but would not be an optimal one. Preference constraints

can often be encoded as costs on individual variable assignments—for example, assigning an afternoon slot for Prof. X costs 2 points against the overall objective function, whereas a morning slot costs 1. With this formulation, CSPs with preferences can be solved using opti

The preceding section gave a formulation of CSPs as search problems. Using this formulation, any of the search algorithms from Chapters 3 and 4 can solve CSPs. Suppose we apply breadth-first search to the generic CSP problem formulation given in the preceding section.

We quickly notice something terrible: the branching factor at the top level is  $nd$ , because any

of  $d$  values can be assigned to any of  $n$  variables. At the next level, the branching factor is

$(n - 1)d$ , and so on for  $n$  levels. We generate a tree with  $n! \cdot d$

n

leaves, even though there are

only d

n possible complete assignments!

Our seemingly reasonable but naïve problem formulation has ignored a crucial property

**COMMUTATIVITY** common to all CSPs: commutativity. A problem is commutative if the order of application

of any given set of actions has no effect on the outcome. This is the case for CSPs be-

cause, when assigning values to variables, we reach the same partial assignment, regardless

of order. Therefore, all CSP search algorithms generate successors by considering possible

assignments for only a single variable at each node in the search tree. For example, at the

root node of a search tree for coloring the map of Australia, we might have a choice between

SA = red, SA = green, and SA = blue, but we would never choose between SA = red and

WA = blue. With this restriction, the number of leaves is d

n

, as we would hope.

The term **backtracking search** is used for a depth-first search that chooses values for **BACKTRACKING**

**SEARCH**

one variable at a time and backtracks when a variable has no legal values left to assign.

The

algorithm is shown in Figure 5.3. Notice that it uses, in effect, the one-at-a-time method of

**AI Adversarial search:** Adversarial search is a **game-playing** technique where the agents are surrounded by a competitive environment. A conflicting goal is given to the agents (multiagent). These agents compete with one another and try to defeat one another in order to win the game. Such conflicting goals give rise to the [adversarial search](#). Here, game-playing means discussing those games where **human intelligence** and **logic factor** is used, excluding other factors such as **luck factor**. **Tic-tac-toe, chess, checkers, etc.**, are such type of games where no luck factor works, only mind works.

Mathematically, this search is based on the concept of ‘**Game Theory.**’ *According to game theory, a game is played between two players. To complete the game, one has to win the game and the other loses automatically.*



### Techniques required to get the best optimal solution

There is always a need to choose those algorithms which provide the best optimal solution in a limited time. So, we use the following techniques which could fulfill our requirements:

- **Pruning:** A technique which allows ignoring the unwanted portions of a search tree which make no difference in its final result.
- **Heuristic Evaluation Function:** It allows to approximate the cost value at each level of the search tree, before reaching the goal node.

### Elements of Game Playing search

To play a game, we use a game tree to know all the possible choices and to pick the best one out.

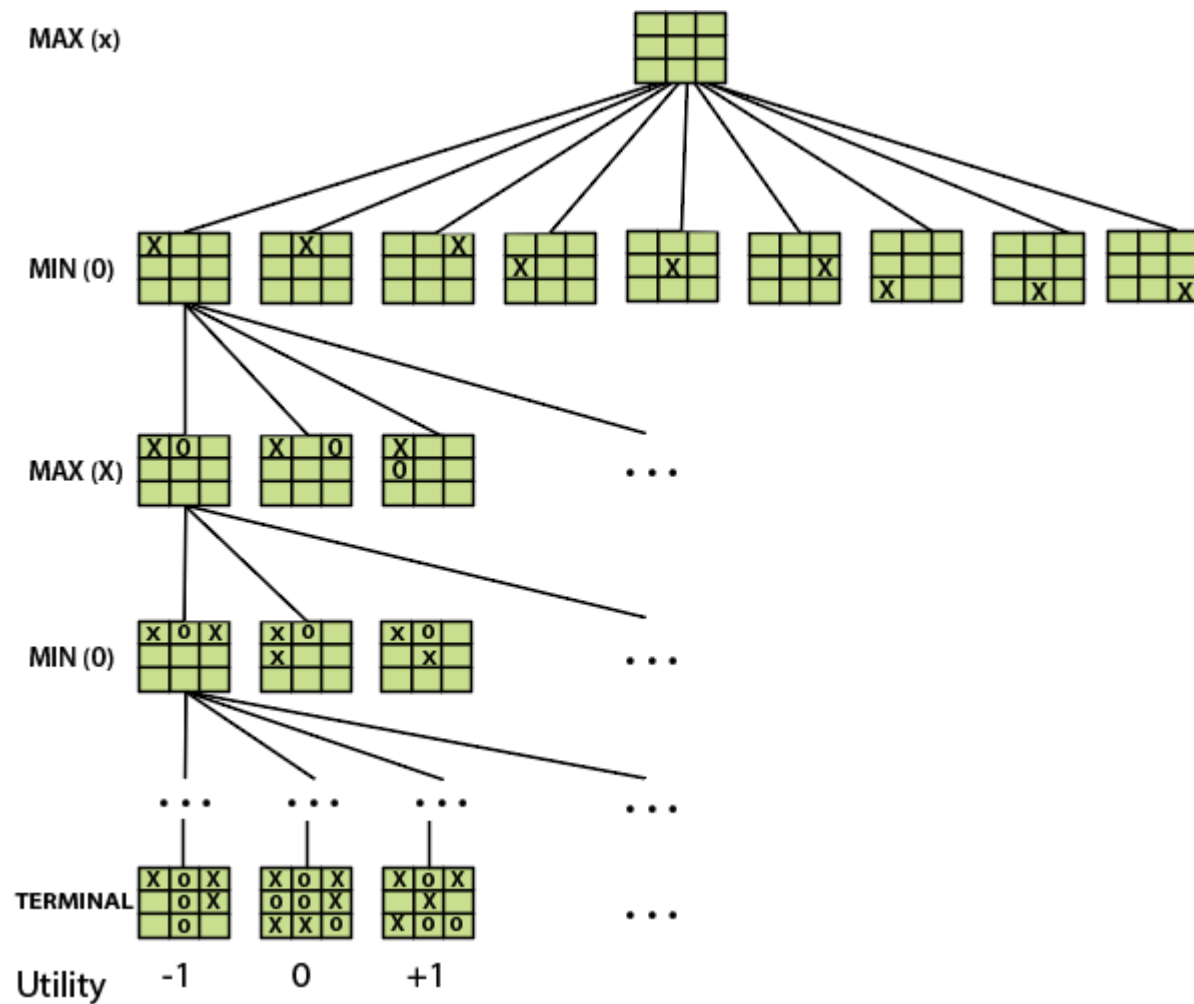
There are following elements of a game-playing:

- **S<sub>0</sub>:** It is the initial state from where a game begins.
- **PLAYER (s):** It defines which player is having the current turn to make a move in the state.
- **ACTIONS (s):** It defines the set of legal moves to be used in a state.

- **RESULT (s, a):** It is a transition model which defines the result of a move.
- **TERMINAL-TEST (s):** It defines that the game has ended and returns true.
- **UTILITY (s,p):** It defines the final value with which the game has ended. This function is also known as **Objective function** or **Payoff function**. The price which the winner will get i.e.
- **(-1):** If the PLAYER loses.
- **(+1):** If the PLAYER wins.
- **(0):** If there is a draw between the PLAYERS.

*For example, in chess, tic-tac-toe, we have two or three possible outcomes. Either to win, to lose, or to draw the match with values +1,-1 or 0.*

Let's understand the working of the elements with the help of a game tree designed for **tic-tac-toe**. Here, the *node* represents the game state and edges represent the moves taken by the players.



### A game-tree for tic-tac-toe

- **INITIAL STATE ( $S_0$ ):** The top node in the game-tree represents the initial state in the tree and shows all the possible choice to pick out one.
- **PLAYER (s):** There are two players, **MAX and MIN**. **MAX** begins the game by picking one best move and place **X** in the empty square box.
- **ACTIONS (s):** Both the players can make moves in the empty boxes chance by chance.
- **RESULT (s, a):** The moves made by **MIN** and **MAX** will decide the outcome of the game.
- **TERMINAL-TEST(s):** When all the empty boxes will be filled, it will be the terminating state of the game.
- **UTILITY:** At the end, we will get to know who wins: **MAX** or **MIN**, and accordingly, the price will be given to them.

### Types of algorithms in Adversarial search

In a **normal search**, we follow a sequence of actions to reach the goal or to finish the game optimally. But in an **adversarial search**, the result depends on the players which will decide the result of the game. It is also obvious that the solution for the goal state will be an optimal solution because the player will try to win the game with the shortest path and under limited time.

There are following types of adversarial search:

- **Minimax Algorithm**
- **Alpha-beta Pruning.**

Minimax Strategy

In artificial intelligence, minimax is a **decision-making** strategy under **game theory**, which is used to minimize the losing chances in a game and to maximize the winning chances. This strategy is also known as '**Minimax,**' '**MM,**' or '**Saddle point.**' Basically, it is a two-player game strategy where *if one wins, the other loose the game*. This strategy simulates those games that we play in our day-to-day life. Like, if two persons are playing chess, the result will be in favor of one player and will unfavor the other one. The person who will make his *best try, efforts as well as cleverness, will surely win.*

We can easily understand this strategy via **game tree**— where the *nodes represent the states of the game and edges represent the moves made by the players in the game*. Players will be two namely:

- **MIN:** Decrease the chances of **MAX** to win the game.
- **MAX:** Increases his chances of winning the game.

They both play the game alternatively, i.e., turn by turn and following the above strategy, i.e., if one wins, the other will definitely lose it. Both players look at one another as competitors and will try to defeat one-another, giving their best.

In minimax strategy, the result of the game or the utility value is generated by a **heuristic function** by propagating from the initial node to the root node. It follows the **backtracking technique** and backtracks to find the best choice. MAX will choose that path which will increase its utility value and MIN will choose the opposite path which could help it to minimize MAX's utility value.

#### MINIMAX Algorithm

MINIMAX algorithm is a backtracking algorithm where it backtracks to pick the best move out of several choices. MINIMAX strategy follows the **DFS (Depth-first search)** concept. Here, we have two players **MIN and MAX**, and the game is played alternatively between them, i.e., when **MAX** made a move, then the next turn is of **MIN**. It means the move made by MAX is fixed and, he cannot change it. The same concept is followed in DFS strategy, i.e., we follow the same path and cannot change in the middle. That's why in MINIMAX algorithm, instead of BFS, we follow DFS.

- Keep on generating the game tree/ search tree till a limit **d**.
- Compute the move using a heuristic function.
- Propagate the values from the leaf node till the current position following the minimax strategy.
- Make the best move from the choices.

#### Alpha-beta Pruning | Artificial Intelligence

Alpha-beta pruning is an advance version of MINIMAX algorithm. The drawback of minimax strategy is that it explores each node in the tree deeply to provide the best path among all the paths. This increases its time complexity. But as we know, the performance measure is the first

consideration for any optimal algorithm. Therefore, alpha-beta pruning reduces this drawback of minimax strategy by less exploring the nodes of the search tree.

The method used in alpha-beta pruning is that it **cutoff the search** by exploring less number of nodes. It makes the same moves as a minimax algorithm does, but it prunes the unwanted branches using the pruning technique (discussed in adversarial search). Alpha-beta pruning works on two threshold values, i.e.,  $\alpha$  (**alpha**) and  $\beta$  (**beta**).

- $\alpha$ : It is the best highest value, a **MAX** player can have. It is the lower bound, which represents negative infinity value.
- $\beta$ : It is the best lowest value, a **MIN** player can have. It is the upper bound which represents positive infinity.

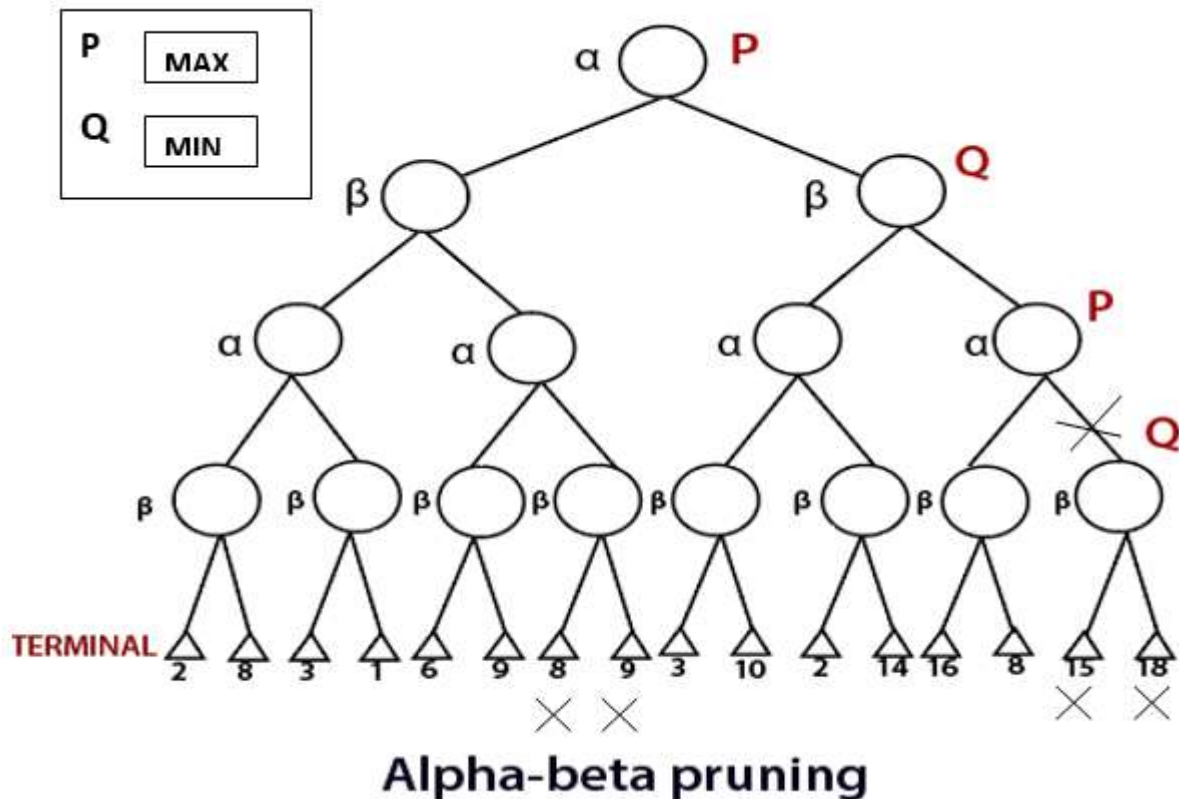
So, each MAX node has  $\alpha$ -value, which never decreases, and each MIN node has  $\beta$ -value, which never increases.

**Note:** Alpha-beta pruning technique can be applied to trees of any depth, and it is possible to prune the entire subtrees easily.

#### **Working of Alpha-beta Pruning**

Consider the below example of a game tree where **P** and **Q** are two players. The game will be played alternatively, i.e., chance by chance. Let, **P** be the player who will try to win the game by maximizing its winning chances. **Q** is the player who will try to minimize **P**'s winning chances. Here,  $\alpha$  will represent the maximum value of the nodes, which will be the value for **P** as well.  $\beta$  will represent the minimum value of the nodes, which will be the value of **Q**.

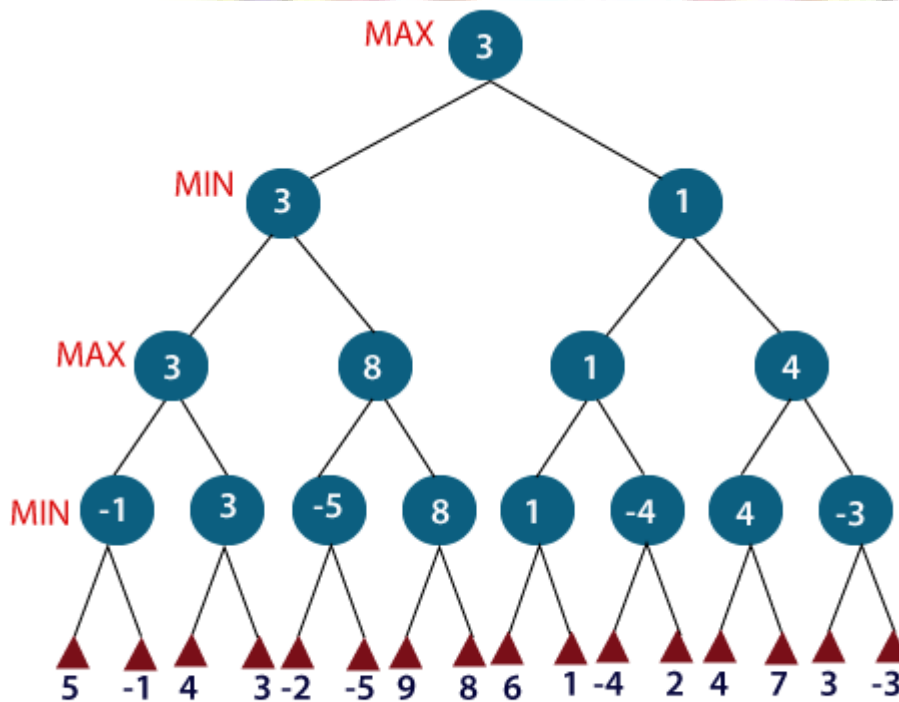




- Any one player will start the game. Following the DFS order, the player will choose one path and will reach to its depth, i.e., where he will find the **TERMINAL** value.
- If the game is started by player P, he will choose the maximum value in order to increase its winning chances with maximum utility value.
- If the game is started by player Q, he will choose the minimum value in order to decrease the winning chances of A with the best possible minimum utility value.
- Both will play the game alternatively.
- The game will be started from the last level of the game tree, and the value will be chosen accordingly.
- Like in the below figure, the game is started by player Q. He will pick the leftmost value of the **TERMINAL** and fix it for beta (?). Now, the next **TERMINAL** value will be compared with the ?-value. If the value will be smaller than or equal to the ?-value, replace it with the current ?-value otherwise no need to replace the value.

- After completing one part, move the achieved  $\alpha$ -value to its upper node and fix it for the other threshold value, i.e.,  $\beta$ .
- Now, its P turn, he will pick the best maximum value. P will move to explore the next part only after comparing the values with the current  $\alpha$ -value. If the value is equal or greater than the current  $\alpha$ -value, then only it will be replaced otherwise we will prune the values.
- The steps will be repeated unless the result is not obtained.
- So, number of pruned nodes in the above example are **four** and MAX wins the game with the maximum **UTILITY** value, i.e.,**3**

The rule which will be followed is: “**Explore nodes if necessary otherwise prune the unnecessary nodes.**”



For example, in the above figure, the two players **MAX** and **MIN** are there. **MAX** starts the game by choosing one path and propagating all the nodes of that path. Now, **MAX** will backtrack to the initial node and choose the best path where his utility value will be the maximum. After this, its **MIN** chance. **MIN** will also propagate through a path and again will backtrack, but **MIN** will choose the path which could minimize **MAX** winning chances or the utility value.

*So, if the level is minimizing, the node will accept the minimum value from the successor nodes. If the level is maximizing, the node will accept the maximum value from the successor.*

## Unit II

**Knowledge representation and reasoning (KR<sup>2</sup>, KR&R)** is the field of [artificial intelligence](#) (AI) dedicated to representing information about the world in a form that a computer system can utilize to solve complex tasks such as [diagnosing a medical condition](#) or [having a dialog in a natural language](#). Knowledge representation incorporates findings from psychology[1] about how humans solve problems and represent knowledge in order to design [formalisms](#) that will make complex systems easier to design and build. Knowledge representation and reasoning also incorporates findings from [logic](#) to automate various kinds of *reasoning*, such as the application of rules or the relations of [sets](#) and [subsets](#).

Examples of knowledge representation formalisms include [semantic nets](#), [systems architecture](#), [frames](#), rules, and [ontologies](#). Examples of [automated reasoning](#) engines include [inference engines](#), [theorem provers](#), and classifiers.

A Knowledge Based Agent in Artificial Intelligence has two levels: Knowledge Base (KB) and Inference Engine.

1. Knowledge Base- It is the base level of an agent, which consist of domain specific content. In this level agent has facts or information about the surrounding environment in which they are working. It does not consider the actual implementation.

2. Implementation level- It consists of domain independent algorithms. At this level, agents can recognize the data structures used in the knowledge base and algorithms which use them. For example, propositional logic and resolution. Knowledge based agents are crucial to use in partially observable environments. Before choosing any action, knowledge based agents make use of the existing knowledge along with the current inputs from the environment in order to infer hidden aspects of the current state.

An agent makes use of TELL and ASK mechanism.

-TELL the agent, about surrounding environment what it needs to know in order to perform some action. TELL mechanism is similar to taking input for a system.

-Then the agent ASKs itself what action should be carried out to get desired output. ASK mechanism is similar to producing output for a system. However, ASK mechanism makes use of the knowledge base to decide what it should do.

*TELL(K): Is a function that adds knowledge K to the knowledge base.*

*ASK(K): Is a function that queries the agent about the truth of K.*

An agent carries out the following operations: First, it TELLS the knowledge base about facts/information it perceives with the help of sensors. Then, it ASKs the knowledge base what action should be carried out based on the input it has received. Then it performs the selected action with the help of effectors.

### **Representation of knowledge in AI**

The objective here is to express knowledge in a computer traceable form such that it can be used to help agents perform well. To make programs capable of representing and using this knowledge in a more explicit way. There is a set of facts which the program will use to figure out how to solve the problem. One approaches to deal with knowledge is based on Logic.

Let's start with the simplest type of logic.

#### **Propositional logic**

It is a simple knowledge representation language. It works at the sentential level. The sentences here are called propositions. We do not go within the individual sentences and discuss their meaning. Propositional logic is unambiguous and is also called Boolean logic as the sentences or propositions return a true or false value. Further, all other logic like First Order Logic are built on top of Propositional Logic. For instance,

*Grass is green.*

*It is Sunday.*

$2 + 2 = 4.$

These are all valid propositions, whereas

*Close the door.*

*Is it Monday today?*

$x = x$  are invalid propositions.

Atomic propositions are simple propositions and Compound propositions are constructed by combining atomic propositions using logical connectives and parenthesis. For instance, It is Sunday and it is a holiday. Logical connectives are used in propositional logic to connect two sentences logically:

Name	Symbol	Stands for
Conjunction	$\wedge$	and
Disjunction	$\vee$	or
Implication	$\rightarrow$	if- then
Negation	$\sim$	not
Biconditional	$\leftrightarrow$	If and only if

Let's look at this with the help of an example. If it is hot and humid, then it is raining.

Here P= It is hot,

Q= It is humid,

R=It is raining

This can be represented as:

$$(P \wedge Q) \rightarrow R$$

### **Properties of Propositional Logic**

1. Commutativity
2. Associativity
3. Distributive property
4. DeMorgan's law
5. Double negation elimination
6. Identity element

However, Propositional Logic has limited expressive power. It cannot be used to represent specializations, generalizations, or patterns. Elements like all, some and none cannot be expressed using propositional logic. For example, All mushrooms are brown, some grapes are sweet, and more.

### **Predicate logic or First Order Logic**

It is another knowledge representation language built on top of propositional logic. It is more expressive as compared to propositional logic and supports temporal information. It is also called higher order logic and can support complex statements.

### **Components**

- User defines primitives:

1. Constant symbols- terms with fixed value which belong to the domain. Example: Tara, 2.

2. Function symbols- mapping individuals to individuals. Example: Mary is the mother of Tara.  
 $\text{mother\_of}(\text{Tara})=\text{Mary}$

3. Predicate symbols- mapping individuals to truth values. Example: 4 is greater than 2.  
 $\text{greater}(4,2)$

- FOL supports these primitives:

1. Variable symbols-  $x, y$

2. Connectives- conjunction, disjunction, implication, negation, biconditional

3. Quantifiers

A. Universal — It stands for “for all” and is represented by the symbol  $\forall$ . It corresponds to conjunction ‘and.’ Example: All dolphins are mammals.

$\forall x: \text{dolphin}(x) \rightarrow \text{mammal}(x)$

B. Existential- It stands for “there exists”. It is represented by the symbol  $\exists$  and corresponds to disjunction ‘or.’ Example: Some mammals lay eggs.

$\exists x: \text{mammal}(x) \rightarrow \text{lay\_eggs}(x)$

C. Equal:  $X=Y$

In predicate logic, a rule has two parts: predecessor and successor. If the predecessor is true, the successor is true. It uses the implication symbol. It represents if-then type of sentences. Example: If the bag is of pink color, I will buy it.  $\text{colour}(\text{bag}, \text{pink}) \text{ --- } \rightarrow \text{buy}(\text{bag})$

### Inference in FOL- Chaining

It is used for simple problems or arrangement of facts. It is used when the knowledge base consists of sentences with horn clauses (I.e. disjunction of literals of which at the most one is positive).

### **Forward Chaining**

It is an inference technique that starts with sentences in the knowledge base and generates new conclusions that can in turn allow more inferences to be made. It is a data driven approach. We start with known facts and arrange or chain them in order to reach the query. It uses modus ponens. It starts with available information and uses inference rules to extract more data until the goal is reached.

Example: To conclude color of the pet named Fritz, given that it croaks and eats flies. The Knowledge Base contains these rules:

If X croaks and eats flies  $\text{ --- } \rightarrow$  Then X is a frog.

If X sings  $\text{ --- } \rightarrow$  Then X is a bird.

If X is a frog  $\text{ --- } \rightarrow$  Then X is green.

If x is a bird  $\text{ --- } \rightarrow$  Then X is blue.

Given: Croaks and eats flies is searched in the KB. This leads us to rule 1 as its antecedent matches our data. The consequent of rule 1 i.e. X is a frog is added to the KB. Now the KB is searched again and rule 3 is chosen since its antecedent (X is a frog) matches our data that was just confirmed. Then the new consequent is added to the KB (i.e. X is green). Thus we have reached the goal of determining color of the pet given than it croaks and eats flies.



## **Backward Chaining**

It is an inference technique that starts from the goal. We find implication sentences that allow us to conclude the it and attempt to establish its premises. It uses modus ponens backwards. It is a goal driven approach and is used in theorem proving.

Example: To conclude color of the pet named Fritz, given that it croaks and eats flies. The KB contains these rules:

If X croaks and eats flies  $\rightarrow$  Then X is a frog.

If X sings  $\rightarrow$  Then X is a bird.

If X is a frog  $\rightarrow$  Then X is green.

If x is a bird  $\rightarrow$  Then X is blue.

The third and forth rules are selected as they match our goal of determining color of the pet. i.e. X is green or X is blue. Both the antecedents of the rules i.e. X is a frog and X is a bird are added to the goal list. Then KB is searched again and the first two rules are selected as their consequents match the new goals added to the list i.e. X is a frog or X is a bird. The antecedent (If X croaks and eats flies) is true/ given and thus we can conclude than Fritz is a frog. The goal of determining color of the pet is achieved ( Fritz is green if it is a frog and blue if it is a bird. But it is a frog as it croaks and eats flies). Thus Fritz is Green.

## Resolution

It is a type of inference technique. The following steps are executed.

1. Convert data into logical statements (propositional or predicate logic).
2. Convert these into Conjunctive Normal form (CNF).
3. Negate the conclusion.

4. Resolve using resolution tree.

Example: If the maid stole the jewellery, then the butler was not guilty.

Either the maid stole the maid stole the jewellery, or she milked the cow.

If the maid milked the cow, then the butler got the cream.

Therefore, if the butler was guilty, then he got his cream.

Step 1 : Expressing as propositional logic.

P= maid stole the jewellery.

Q= butler is guilty.

R= maid milked the cow.

S= butler got the cream.

Step 2: Convert to propositional logic.

1.  $P \rightarrow \sim Q$

2.  $P \vee R$

3.  $R \rightarrow S$

4.  $Q \rightarrow S$  (Conclusion)

Step 3: Converting to CNF.

1.  $\sim P \vee \sim Q$

2.  $P \vee R$

3.  $\sim R \vee S$

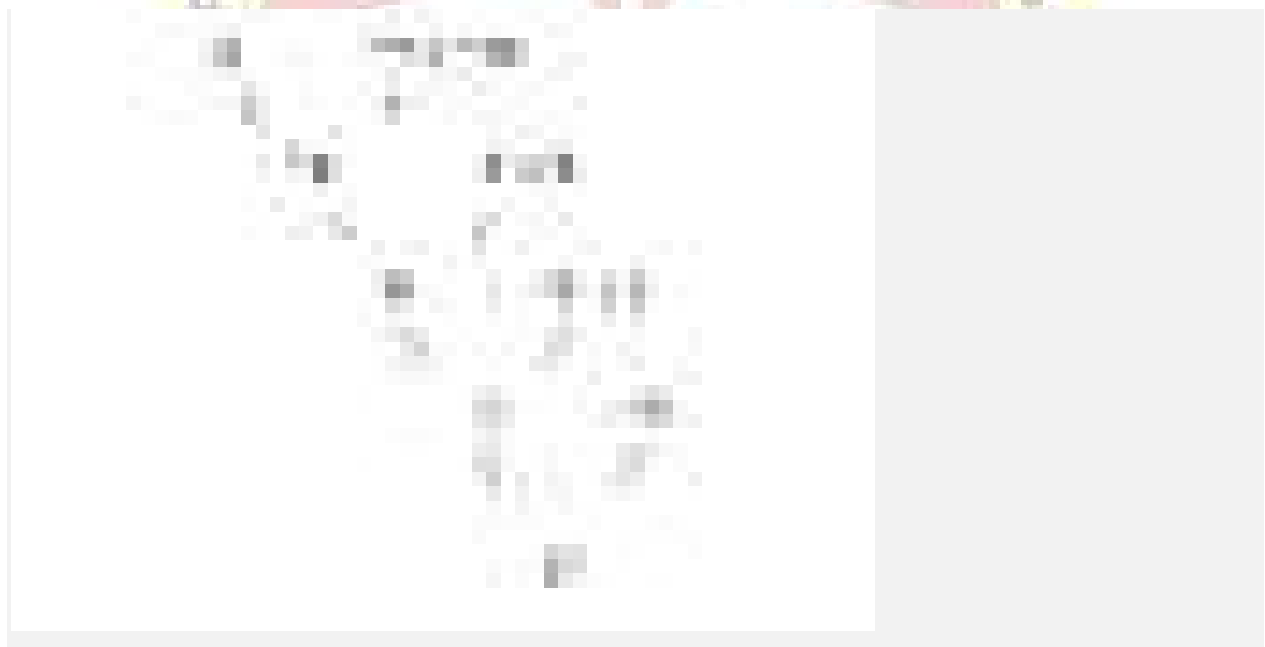
4.  $\sim Q \vee S$  (Conclusion)

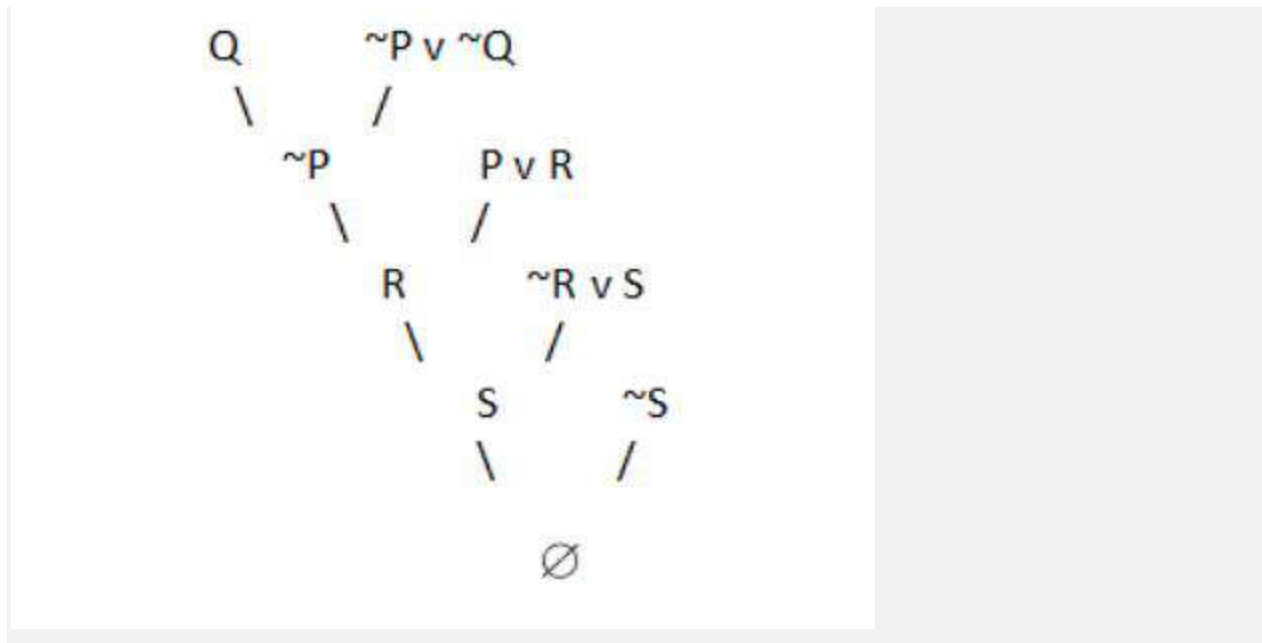
Step 4:

Negate the conclusion.

$$\sim(\sim Q \vee S) = Q \wedge \sim S$$

It is not in CNF due to the presence of '^'. Thus we break it into two parts:  $Q$  and  $\sim S$ . We start with  $Q$  and resolve using resolution tree.





Negation of the conclusion gives a null value. Hence our conclusion is proved.

### *Semantic Networks*

This document concerns the management of the output of insight generators, the software agents utilized in the insight generation systems. The solution to managing these reports involves the automatic creation of a repository for all materials generated by various insight generators; this repository allows the user to navigate through this continually growing space of marketing reports, gaining new insights about the relationships between items of interest and adding new insights in the process. The goal of the system is to make all marketing information and insights generated by the man/machine interaction available to the user, so that there is a convergence towards a "conservation of information". To use a geometric metaphor, the goal is to make the user equidistant from all information at all times, as illustrated below.

The output of insight generators like [I Want](#) is paper; every time the system is run, a paper report is produced. If the system were run for every retail account in every market, it would produce hundreds of reports. If similar insight generators focused on other insights (distribution, variety, coupons, shelf space, prices, etc.), the collection of agents would produce thousands of reports. If each agent were run for each brand item (each size, package type, flavor, etc.), there could be

many thousands of reports. Finally, if all of the agents were run each month for all the brands, then millions of reports could accumulate.

Further, these agents could be run for a firm's competitors. The system could be run *backwards*; that is to say, it could be run from the perspective of the brand's competitors in a particular category. From this, the brand group could learn where their brand is vulnerable to attack from competitors seeking to take merchandising support away from it. Also, the sales force could be informed where *not* to ask for more support, such as in the instance where their brand is receiving far more feature support than its volume share warrants. Such use of this system can be called *exposure analysis* and it was further explored in the Market Opportunity Inspector (MOI) prototype.

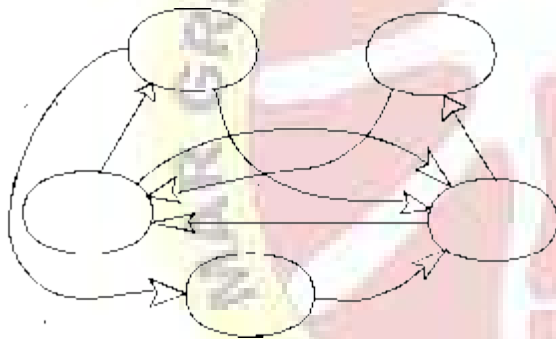
One approach to dealing with all of these insights is summarization, which is explored in the Marketing Opportunity Inspector (MOI) document. The MOI system assumes that an "I Want..." system has been run for a brand in all accounts in all markets. Each page of output corresponds to an exposure, an opportunity or neither for the brand. MOI takes these and summarizes the number and magnitude of the opportunities and exposures. The output of the system is again a sheet of paper. MOI helps to generate insights about a brand's situation, by locating its strengths and its weaknesses. The output of one insight generation system (I Want) became the input of another insight generating system (MOI). In both systems the output is a high-quality paper report. Insight generating systems such as MOI are employed to summarize the lower level information and pinpoint insights from this mass of textual output. But, the summaries themselves also add to the ever-growing output and must also be made available to users.

A second approach to the problem of insight management is to manage the output itself as a sort of *library* of information known about the brand. This library would be part of a system that manages information and insights that are in the form of compound documents, i.e. pictures and text, that goes beyond a strict hierarchical arrangement. The user needs to be able to move in multiple directions from any vantage point and create hierarchies as needed. The information needs to be arranged in a structure that follows the intrinsic relationships between the context of the reports and their contents. The reports must be managed according to some "mental-model" of the world of marketing.

One technology for capturing and reasoning with such mental models is a semantic network ... the topic of this document.

What is a Semantic Network?

Semantic networks are knowledge representation schemes involving nodes and links (arcs or arrows) between nodes. The nodes represent objects or concepts and the links represent relations between nodes. The links are directed and labeled; thus, a semantic network is a directed graph. In print, the nodes are usually represented by circles or boxes and the links are drawn as arrows between the circles as in Figure 1. This represents the simplest form of a semantic network, a collection of undifferentiated objects and arrows. The structure of the network defines its meaning. The meanings are merely which node has a pointer to which other node. The network defines a set of binary relations on a set of nodes.



**Figure 1**

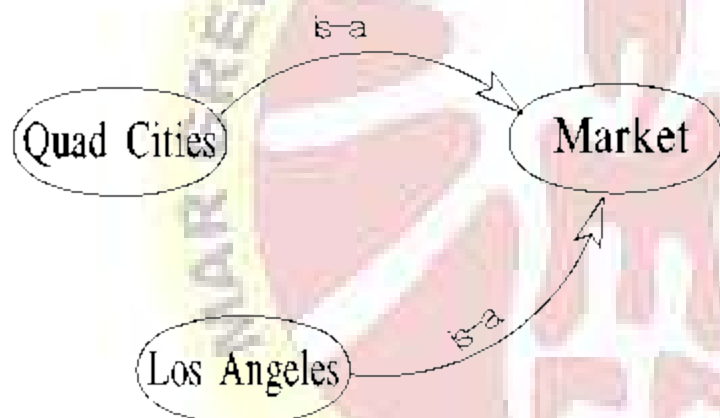
Pick up almost any technical book and look in the preface or introduction. Invariably there is a chapter dependency diagram. It is a node-link structure, a semantic network in which the nodes represent chapters and the links represent the relationship of which chapters should be read before which other chapters.

To move semantic nets from this abstract realm to something more concrete, let us consider an example from the structure of marketing. To begin simply, let us introduce two nodes and a link.



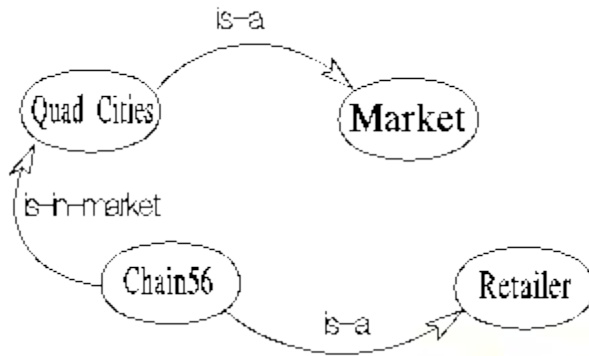
**Figure 2**

The node on the left labeled "Quad Cities" is linked to the node on the right, labeled "Market", and the arrow is labeled "is-a". Quad Cities is an example of a market. The diagram, in other words represents the fact that there is a binary relation between a market, Quad Cities, and the concept of a market. Another node with the label "Los Angeles" and a "is-a" link from this node to the "Market" node could be added, again representing that "Los Angeles" is a type of "Market".



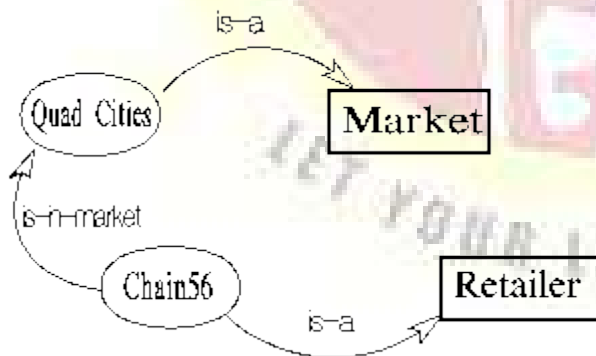
**Figure 3**

If a retailer node is added to Figure 2, the structure of the network becomes apparent as shown in Figure 4. Markets generally contain retailing entities. To add an example of a retailer, add a node labeled "Chain56" and two links - one from the retailer "Chain56" to "Quad Cities" labeled "is-a-retailer-in" and one from the node "Chain56" to the node "Retailer" labeled "is-a". This illustrates that Chain 56 is a retailer in the Quad Cities market.



**Figure 4**

It is now important to note a point or two of possible semantic confusion. Notice that the nodes in this small network are not all of the same "type". The node labeled "Market" represents the generic or meta or class concept of a market; it represents the abstract concept of a market. It can be thought of as possessing properties common to all markets. The node "Quad Cities" represents an individual instance of the node "Market". The node "Quad Cities" represents a particular market. The same is true of the relation between the node labeled "Retailer" and the node labeled "Chain56". The node "Retailer" again represents the concept of a retailer that is common across all particular retailers. One instance of such a retailer is the node labeled "Chain56". In order to distinguish between these two types of nodes, the class nodes become boxes and the instance nodes become ellipses, as in Figure 5.

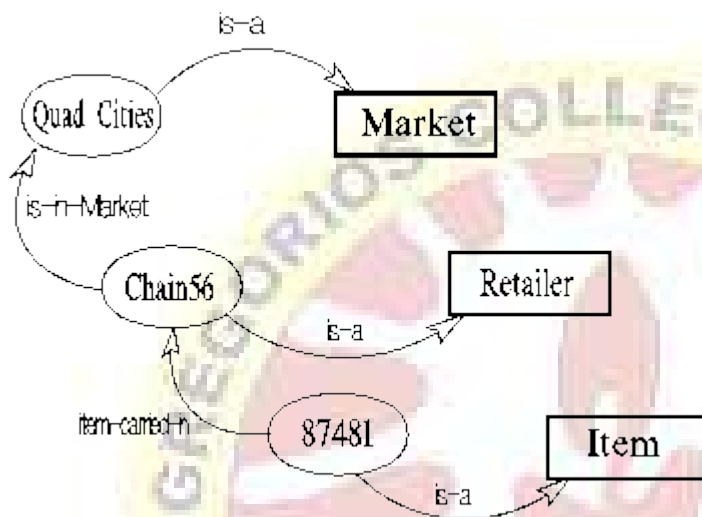


**Figure 5**

Another class node, labeled "Item", that represents the abstraction of items in a category, can now be added. Along with that, an instance of an item, labeled "87481", is added. Notice that there is a strong relationship between the type or class nodes and the column headings or entities



of a relational database table. We will exploit this similarity later in this paper. Thus, another "is-a" link and a new link, "item-carried-in", must be added to the node "87481" and the node "Chain56" respectively. These new additions are shown in Figure 6. The information now being represented is that Chain56 is a retailer in the Quad Cities market and that Chain56 carries the item 87481.



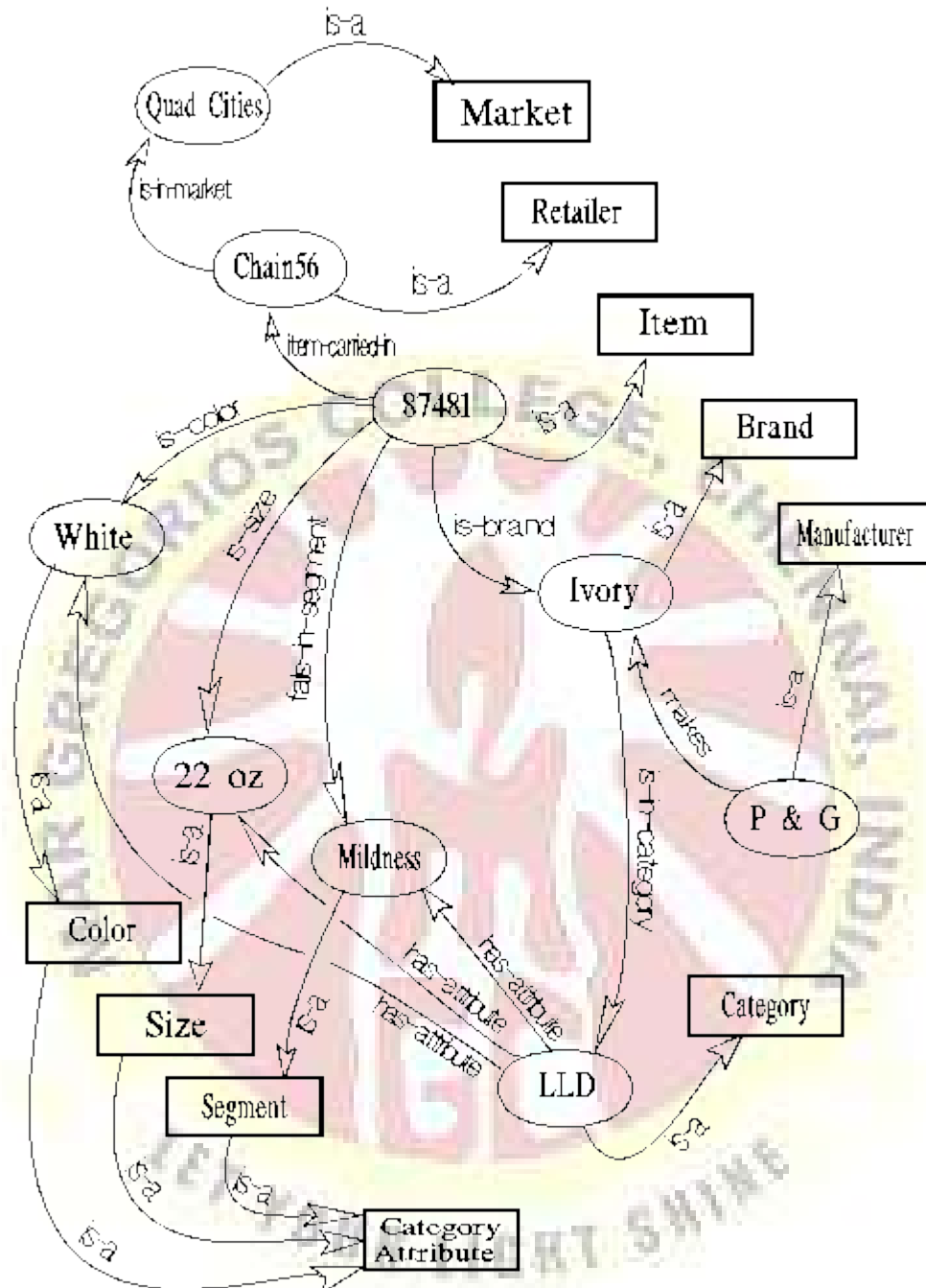
**Figure 6**

As the nodes proliferate, the meanings of these links need to be considered. It should become apparent that not all links are alike. Some links express only relationships between nodes, and are therefore "assertions" of the nature of the relationship between two different nodes. For example, the link "item-carried-in" in Figure 6, which illustrates the relationship that retailer Chain56 carries the item 87481. The "is-a" links in Figure 6 are "structural" links in that they convey "type" information about the node. This information is about the node itself and not about the relationship it has to a different "type" of node. For instance, the node labeled "87481" is an instantiation of the class node labeled "Item".

In Figure 7, more nodes and links are introduced to the original network. There is now a "Brand" class node with an instance node "Ivory". The link "is-brand" conveys the information that the item 87481 is the Ivory brand. There are now also class nodes labeled "Manufacturer", "Category", and "Category Attributes". The Category Attributes class is linked to three other class nodes labeled "Size", "Color" and "Segment". These represent particular attributes of a

particular category; in this instance, the liquid light duty detergent category, of which Ivory is a member. The "is-a" links between the class node Category Attributes and the class nodes Size, Color, and Segment represent a relationship of class to subclass and, hence, "structural" links. Here again are links that do not denote a relationship between different types of instance nodes, but give information about a class node itself. The class node Color is a type of Category Attribute.



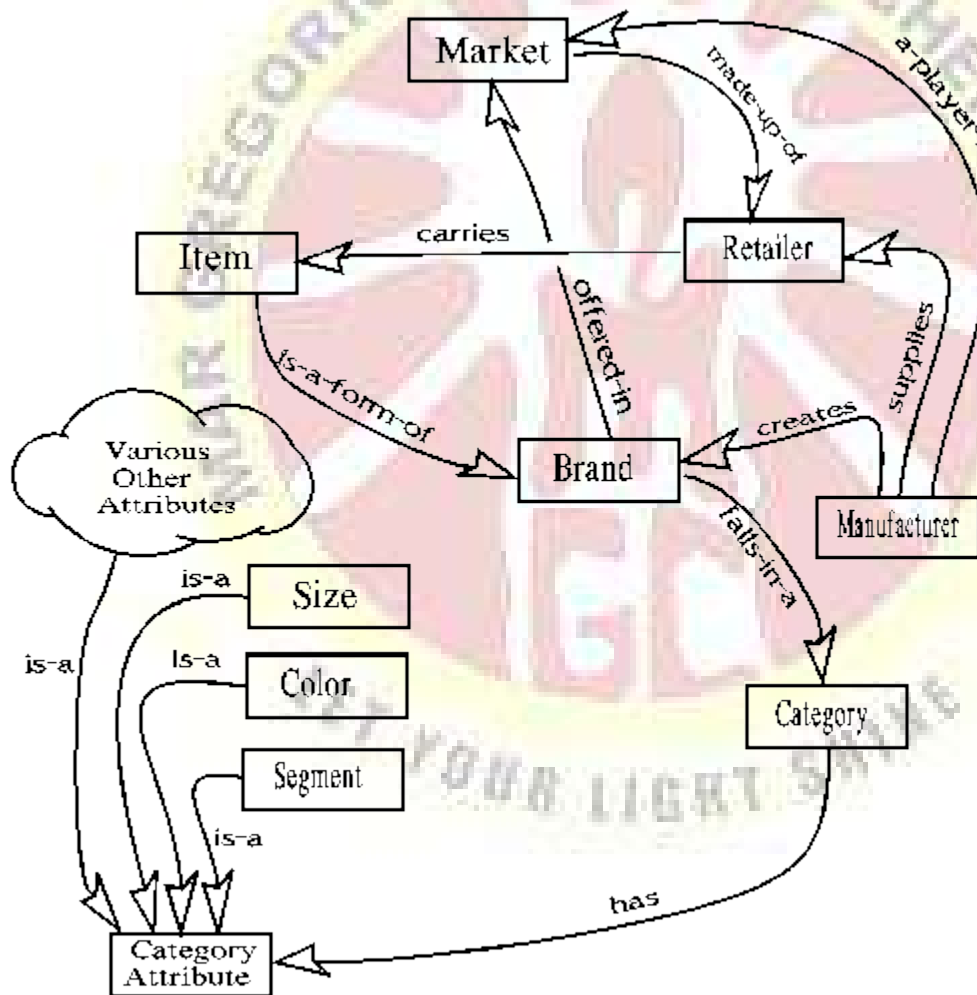


**Figure 7**

Our network in Figure 7 now has a representation for information about the item node 87481. For instance, it is a form of Ivory which is manufactured by Procter & Gamble; it is the 22 ounce size, white in color and competes in the Mildness market segment of the liquid light-duty detergent category. This is one item in one chain in one market. The database used in the Marketing Information Center prototype has 100 items in five chains in one market. Each item

can be one of seventeen brands made by twelve manufacturers which comes in seven sizes and eight colors and can compete in one of five segments. This database is a pared-down version of a scanner database for the Quad Cities market which has many more retailers and a good many more items. The network in Figure 7 becomes very complex with a 100-fold increase in the amount of information.

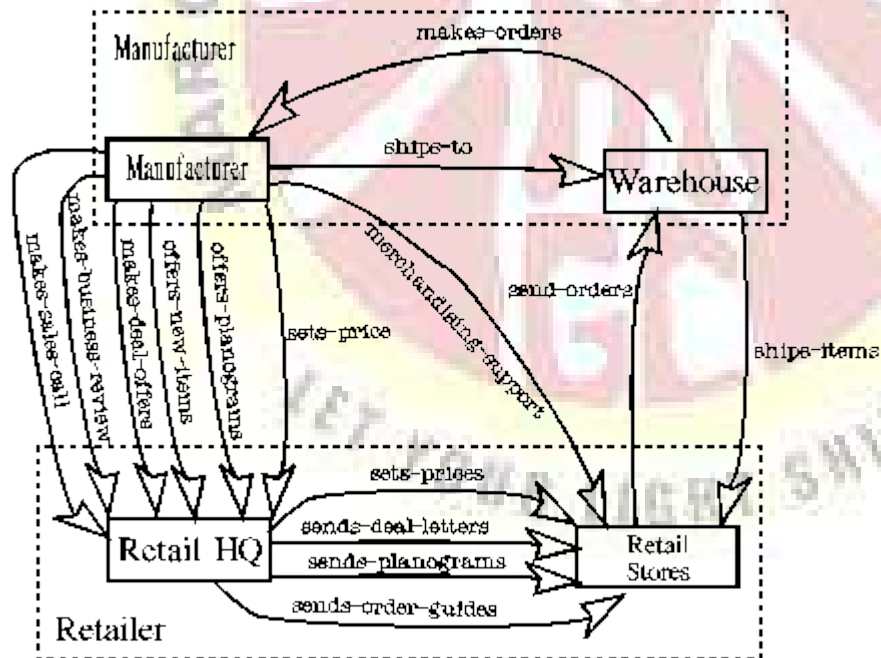
None of the networks have shown any structural links among class nodes, except for Figure 7 which shows only a subclass relationship between Category Attributes nodes and various class node attributes. Figure 8 shows possible structural relationships between class nodes.



**Figure 8**

In this figure the instance nodes have been left out in order to show more clearly possible relationships between classes. Remember class nodes represent larger, more general concepts

and just as general concepts can have more refined sub-concepts, the particular types of category attributes, such as Size, are represented as a sub-class of the broader concept of Category Attributes as shown previously. Notice that the class Category Attributes is a kind of abstract class that probably would never have an instance node tied directly to it. It can only have relationships to other class nodes. So, general concepts are represented such as the concept that there are Manufacturers who create things termed Brands which are supplied to things called Retailers. Retailers are in an abstraction called a Market and carry instances of Items. All of this is obvious from the diagram. What is not so obvious is that the nodes themselves can contain more than meets the eye. The Retailer node is a short-hand notation for a bundle of concepts that make up a real-live retailer, such as the fact that retailers have a headquarters and stores and control shelf space, price and display. The links, such as the one labeled "supplies" between the Manufacturer node and the Retailer node, are really more like a co-axial link than a simple arc. This particular link represents a bundle of various relationships between Manufacturer and the abstraction, Retailer. This detail is shown in Figure 9.



**Figure 9**

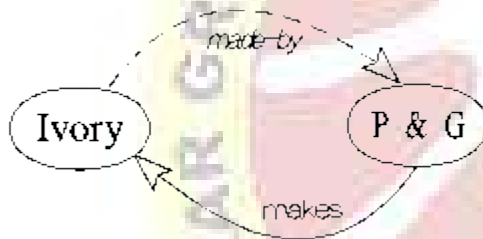
Another important characteristic of the node-link representation is the implicit "inverse" of all relationships represented by the directional arrows. If there is an arrow going from one node to

another, this also implies the reverse - that there is an arrow from the second node to the first. In Figure 10, there are the nodes labeled "P & G" and "Ivory" with the link labeled "makes". The direction of the relationship is that "P & G makes Ivory". Further, some linguistic terminology for our binary relationships could be used: "P & G" is the *subject* and "Ivory" is the *object*, and "makes" is the verb or action or *link* between them. This will be discussed in greater detail later.



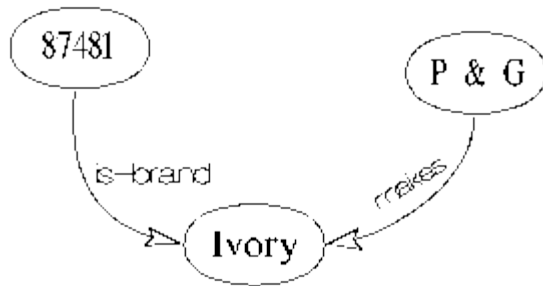
**Figure 10**

This "P & G makes Ivory" relation implies the inverse relationship that "Ivory *is-made-by* P & G", as shown in Figure 11.



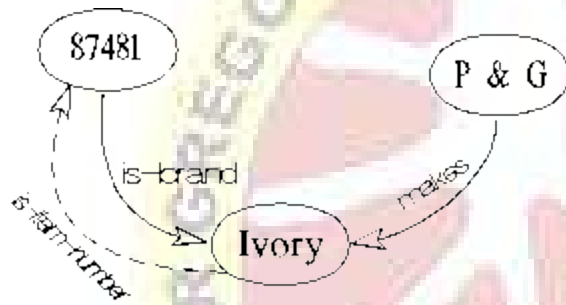
**Figure 11**

The representational or expressive power of semantic networks has been discussed thus far. As with any kind of knowledge representation scheme, a way of inferring knowledge that is not directly represented by the scheme is needed. The ability to work with incomplete knowledge sets a knowledge representation apart from a database. To give an example of what can be gleaned from the semantic network in Figure 7 that is not directly represented, consider Figure 12. It is an extraction of Figure 7 containing only three nodes and two links.



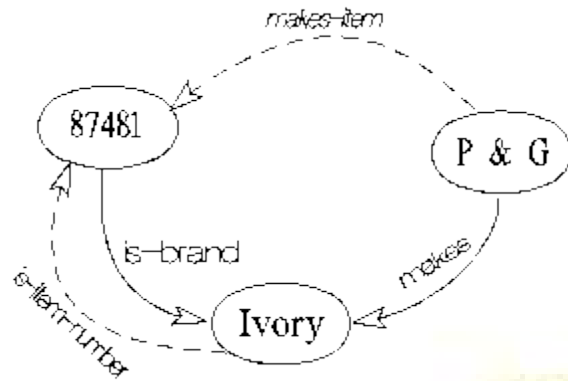
**Figure 12**

The information explicitly represented is that the item numbered 87481 is the Ivory brand and that Procter & Gamble makes Ivory. The inverse relationship of the item 87481 to the brand Ivory, i.e. that Ivory *is-item-number* 87481 is shown in Figure 13.



**Figure 13**

By tracing the path from the node P & G to the node Ivory via the arrow labeled "makes" and then from the node Ivory to the node 87481 via the arrow labeled "is-item-number", we can infer that Procter & Gamble manufactures the item 87481 by inferring a link labeled "makes-item" between the node P & G and the node 87481, as shown in Figure 14. This may seem obvious, but remember this small amount of new information need not be explicitly represented in the original network.

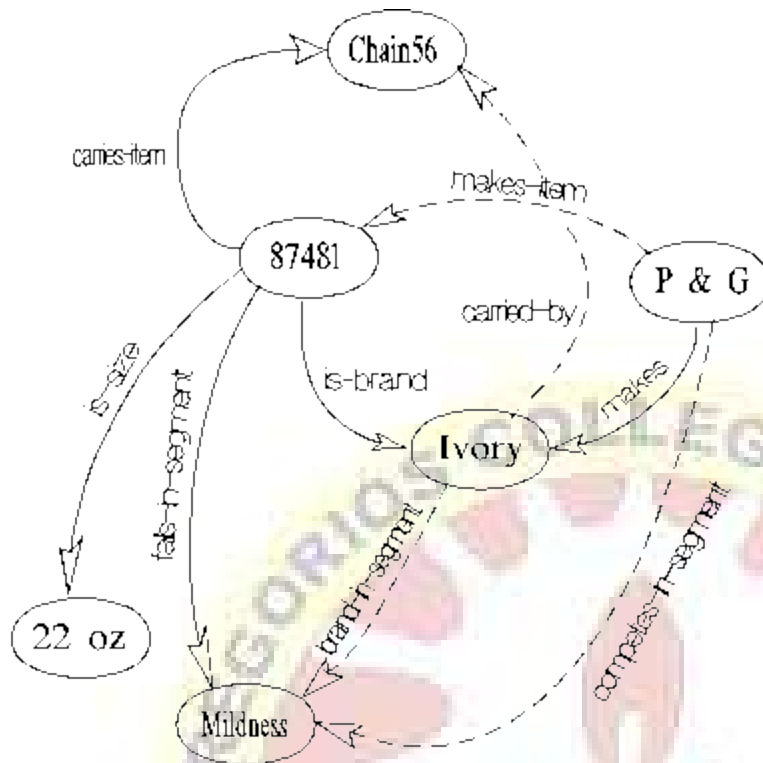


**Figure 14**

Described mathematically, *composing* arrows occurs by placing them end-to-tail. This *composition* creates a new arrow. In Figure 14, a triad of nodes is formed by arrows said to "commute". It is not possible to compose every pair of arrows, only those whose destinations and sources correspond. The destination of the first must be the source of the second. By composing arrows, new relationships between nodes can be found and described. This process is sometimes called "chasing arrows" and the terminology introduced stems from a branch of mathematics called Category Theory.

Figure 15 shows the results of more "arrow chasing". Additional relationships are derived, such as Ivory is a Brand carried by Chain 56, Procter & Gamble makes a product that competes in the Mildness segment, and Ivory is a Brand competing in the Mildness segment. Notice that layers of relationships between nodes can be built.





**Figure 15**

This discussion has introduced the concept of a semantic network consisting of nodes and links. The nodes represent concepts and the links represent relationships between these concepts. A distinction was made between instance nodes and class nodes: the former represents general notions of the latter of which there may be many types. The concept of links which extend from the instance node level to the class node level was given along with an introduction of the notion of abstract classes. The reversibility of the arrows and the method of inferring new relationships between nodes from existing ones was also given. Several figures illustrated these concepts using an example semantic network built from a scanner database header file of the liquid light-duty detergent category.

### **Limitations of Semantic Networks**

This chapter should not end without some discussion of the limitations of semantic networks, and a comparison of their traditional use versus their use in the management of marketing insights.

Semantic networks as a representation of knowledge have been in use in artificial intelligence (AI) research in a number of different areas. Some of the first uses of the nodes-and-links formulation were in the work of Quillian and Winston, where the networks acted as models of associative memory. Quillian's work centers on how natural language is understood and how the meanings of words can be captured in a machine. Winston's work concentrates on machine learning and specifically on structural descriptions of an environment. Winston's work describes pedestals and arches formed from more elementary pieces such as wedges and blocks; these make up the famous "blocks world" that has been utilized by many research efforts in semantic networks.

The other major area in which the use of semantic networks is prevalent, is in models based on linguistics. These stem in part from the work of Chomsky. This latter work is concerned with the explicit representation of grammatical structures of language. It is opposed to other systems that tried to model, in some machine-implementable fashion, the way human memory works. Another approach combining aspects of both the previously mentioned areas of study was taken by Schank in his conceptual dependency approach. He attempted to break down the surface features of language into a network of deeper, more primitive concepts of meaning that were universal and language independent.

Such creations and uses of semantic networks have led to any number of epistemological problems. Numerous researchers have attempted to address these problems. Barr and Feigenbaum state that:

*In semantic network representations, there is no formal semantics, no agreed-upon notion of what a given representational structure means, as there is in logic, for instance.*

Of course, the success of logic in this respect is debatable, but semantic networks do tend to rely upon the procedures that manipulate them.

For example, the system is limited by the user's understanding of the meanings of the links in a semantic network. As pointed out previously in the example of the network of marketing insights, links between nodes are not all alike in function or form. Hence we need to differentiate between links that assert some relationship and links that are structural in nature). A paper by

Brachman on the subtleties of the "is-a" link revealed even more distinctions in the uses of this link. According to Brachman "is-a" links can be divided into two groups, depending upon the nodes involved. One use of the "is-a" link resembles our distinction between an instance node and a class node. The link represents the relationship of instance nodes to abstract, generic qualities shared by many instance nodes. Brachman's other use of a link is between two instance nodes. These two major divisions can then be further broken down into finer uses of the link.

If problems and sublime uses characterize links, then nodes are not much better. The seeming simplicity of a node that represents a single concept or object in the world is actually fraught with complications. The question "what is a node?" is on par with "what is a market?" As in the discussion of links above, we need to distinguish between nodes that represent some set of objects and nodes that represent classes of qualities shared by these objects. There can be properties for instances of a class that all members of a class share as well as properties of the class itself. A property of a market is that it is made up of retailers. For example, a member of the class Market, such as Quad Cities, contains specific retailers, such as Chain 56. Chain 56 is an instance of the class Retailer. The problems of epistemology and the semantics of semantic network representations are discussed further in Brachman.

It should be noted that the sample network discussed at length in this book shares the advantages and disadvantages of any semantic network. We take "naive semantic networks" aka "naive set theory". We do not worry overly about the epistemologico-semantic problems associated with the use of the representation. Some of the difficulties are in fact side-stepped since the final goal for this network is not quite the same as the goals of other researchers. One fundamental difference between the use of the network representation in this book as opposed to elsewhere, is that we are not trying to represent natural language or associative memory independent of users. In fact we rely heavily on the user's intuitive understanding of the world. Semantic networks are used in this project as structures of knowledge that encourage the user to interact with them.

Having marketing knowledge visible to the marketing professional is one of the major advantages of AI technologies and of tools such as spreadsheets. This topic is pursued in more depth in Chapter 18, while the next chapter describes a computer-based tool for implementing semantic networks.

**Abstract.** People communicate with each other in sentences that incorporate two kinds of information: propositions about some subject, and metalevel *speech acts* that specify how the propositional information is used — as an assertion, a command, a question, or a promise. By means of speech acts, a group of people who have different areas of expertise can cooperate and dynamically reconfigure their social interactions to perform tasks and solve problems that would be difficult or impossible for any single individual. This paper proposes a framework for intelligent systems that consist of a variety of specialized components together with logic-based languages that can express propositions and speech acts about those propositions. The result is a system with a dynamically changing architecture that can be reconfigured in various ways: by a human knowledge engineer who specifies a script of speech acts that determine how the components interact; by a planning component that generates the speech acts to redirect the other components; or by a committee of components, which might include human assistants, whose speech acts serve to redirect one another. The components communicate by sending messages to a Linda-like blackboard, in which components accept messages that are either directed to them or that they consider themselves competent to handle.

In the years since its founding conference in 1956, the field of artificial intelligence has generated an impressive collection of valuable components, but no comparably successful architecture for assembling them into intelligent systems. As examples, the following list illustrates the range of AI components that were designed and implemented in the 1950s and '60s:

parsers, theorem provers, inference engines, search engines, learning programs, classification tools, statistical tools, neural networks, pattern matchers, problem solvers, planning systems, game-playing programs, question-answering systems, dialog managers, machine-translation systems, knowledge acquisition tools, modeling tools, and robot guidance systems.

Over the past 40 years, the performance, reliability, and generality of these components have been vastly improved. Their theoretical foundations are much better understood, and they have found their way into applications that are no longer considered part of AI. Yet despite attempts to

integrate the components into general-purpose intelligent systems, the results are disappointing: the commercially successful systems are limited to special-purpose applications, and the more general systems have not progressed beyond the stage of clever demos. Nothing remotely resembling the HAL computer in the movie *2001* exists today, and there are no credible designs for building one soon.

The lack of progress in building general-purpose intelligent systems could be explained by several different hypotheses:

1. Simulating human intelligence on a digital computer is impossible.
2. The ideal architecture for true AI has not yet been found.
3. Human intelligence is so flexible that no fixed architecture can do more than simulate a single aspect of what is humanly possible.

Many people have presented strong, but not completely convincing arguments for the first hypothesis (Winograd & Flores 1986; Penrose 1989; Dreyfus 1992). In the search for an ideal architecture, others have implemented a variety of at best partially successful designs. The purpose of this paper is to explore the third hypothesis: propose a flexible modular framework that can be tailored to an open-ended variety of architectures for different kinds of applications. The tailoring could be done either by a human knowledge engineer who uses specialized AI languages or by semiautomated design tools in collaboration with a human editor who has little or no training in AI. Such a system would not be as intelligent as HAL, but it should be valuable for a wide range of important applications.

The idea of a flexible modular framework (FMF) is not new. It is, in fact, the underlying philosophy of the Unix operating system and its descendants. That philosophy is characterized by four design principles:

1. A small kernel that provides the basic services of resource allocation and process management.
2. A large, open-ended collection of highly modular utilities, which can be used by themselves or be combined with other modules.

3. Glue languages, also called scripting languages, for linking modules to form larger modules or complete applications.
4. A uniform data representation, based on character strings, which constitute the storage format of Unix files and the content transmitted by Unix pipes.

The first three principles are as valid today as they ever were, but the fourth has been modified to accommodate modules that require data with more structure than linear strings, especially database management systems (DBMS) and graphical user interfaces (GUIs). Unix systems implement the DBMS and the GUI as independent modules, but their nonlinear data structures cannot be communicated via pipes. Other operating systems make different compromises: the IBM AS/400 implements the DBMS in the kernel, and the Macintosh and Windows systems implement the GUI in the kernel.

The LISP language, which was the primary language of AI since the late 1950s, pioneered techniques that entered the mainstream of commercial computing when they were adopted by other languages ranging from PL/I to Java. For AI systems, LISP served as both an implementation language and a glue language for AI components and complete systems. Unlike the Unix character strings, the basic data structures of LISP consist of tree-like lists, which can be supplemented with cross links to form arbitrary graphs. During the 1970s and '80s, the trees and graphs of LISP proved to be rich enough to support the operating systems of the LISP machines with their stunning graphics. Those graphics techniques, which were invented at Xerox PARC, have been copied in all modern GUIs, including those of the Macintosh and Windows.

Although LISP was, and still is, a highly advanced programming language, it is not by itself a knowledge representation language. The Prolog language is a step closer to a KR language. It supports the same kinds of data structures as LISP, but it has a built-in inference engine for the Horn-clause subset of logic, which can be used to express the rules of an expert system or the grammars of natural languages. For many applications, Prolog has been used as a KR language, either directly or with some syntactic sugar to make its notation more palatable. Yet Prolog still has limitations that make it unsuitable as the glue language for intelligent systems: procedural dependencies, a nonstandard treatment of negation, and the limited expressive power of Horn-

clause logic. Like LISP, Prolog is better suited to implementing the components of intelligent systems rather than representing the knowledge they process.

The most promising candidate for a glue language is Elephant 2000, which McCarthy (1989) proposed as a design goal for the AI languages of the new millennium. Sentences in the Elephant language include "*requests, questions, offers, acceptances of offers, permissions* as well as *answers to questions* and other assertions of fact. Its outputs also include *promises* and statements of *commitment* analogous to promises." As an inspiration for Elephant, McCarthy cited the *speech acts* of natural languages (Austin 1962; Searle 1969), but he believed that Elephant sentences should be written in a formally defined version of logic, rather than the much more informal natural languages. Unix only supports one kind of speech act: a command that invokes some program. The Unix scripting languages add loops and conditionals, which determine the sequence of commands to execute. Prolog supports two kinds of speech acts: assertions for stating facts and goals for issuing commands or asking questions. Besides those special cases, Elephant provides a framework that can support the full range of illocutionary and perlocutionary speech acts.

Although the glue language for intelligent systems should be at a higher level than the scripting languages of Unix, the four design principles of the Unix philosophy can serve as guidelines. Following are AI generalizations of the four principles:

1. Like the Unix kernel, an AI kernel must support resource allocation and process management. But unlike Unix, which invokes a specific module for each command, an AI kernel should have a pattern-directed or associative method for determining when a module should be invoked. In many AI systems, a *blackboard* or *bulletin board* is used to post messages, which any appropriate component can access when it detects a characteristic pattern. The Linda language (Carriero & Gelernter 1992) is an example of an efficient blackboard system that has been widely used for scheduling parallel computations by clusters of computers.
2. Like Unix, an AI system should have an open-ended collection of modular components, but the kinds of components should be traditional AI tools of the kinds that have been developed over the past 40 years. New kinds may also be needed, but they could also be

invoked by a glue language like Elephant in combination with a Linda-like blackboard. More conventional components, such as a DBMS, GUI, and various networks, could be invoked by the same mechanisms. The Jini system of Java, for example, uses a version of the Linda operators for invoking components distributed across a network.

3. An AI glue language, as McCarthy emphasized, should be based on a version of logic that is rich enough to include all of first-order logic plus metalevels that can talk about the object level and state whatever speech act is intended. Two such languages are conceptual graphs (CGs) and the Knowledge Interchange Format (KIF), which are being standardized as logically equivalent notations for the same model-theoretic foundations. Other versions of logic, which are discussed in Section 5 of this paper, can be translated to or from subsets of CGs and KIF. For communication with people who are not logicians, those logics can also be translated to or from versions of controlled natural languages (CNLs), which can serve as readable notations for the underlying logic.
4. Instead of the linear character strings stored in files and transmitted by pipes, logic provides a much richer notation that can represent all the data structures needed for a DBMS, GUI, or network protocol. The messages posted to a Linda-like blackboard could include any logical expression, which in extreme cases might represent an arbitrarily large graph or even the conjunction of any or all the data in a DBMS. The metalevel capabilities of logic, which can represent any desired speech act, can state what should, would, could, or must be done with the data.

This paper shows how a framework based on these four principles can support a family of architectures that can easily be tailored for different kinds of applications. Section 2 discusses three logically equivalent notations for an Elephant-like glue language: controlled natural language for the human interface; conceptual graphs for components that use graph-based algorithms; and KIF for components that use other notations for logic. Section 3 surveys the use of CNLs as a front end to AI systems. Section 4 shows how graph algorithms can simplify or clarify the techniques for searching, querying, and theorem proving. Section 5 discusses techniques for handling the computational complexities in different applications of logic. Finally, Section 6 discusses the kinds of components needed for a flexible modular framework and how a



glue language communicating through a blackboard can be used to combine them, relate them, and drive them.

## 2. Notations for Logic

McCarthy's Elephant language requires a highly expressive version of logic, but he did not propose any particular notation for it. Although various notations — graphical, linear, or NL-like — can express equivalent semantic information, the choice of notation can have a major influence on both the human interfaces and the kinds of algorithms used in the computations. This section illustrates three notations for logic: conceptual graphs, KIF, and controlled natural languages. All three of them can express exactly the same semantics in logically equivalent ways, but they have complementary strengths and weaknesses that make them better suited to different kinds of tasks. Any or all of them could be used in messages passed through a Linda-like blackboard.

For his syllogisms, the first version of formal logic, Aristotle defined a highly stylized form of Greek, which became the world's first controlled natural language. During the middle ages, Aristotle's sentence patterns were translated to controlled Arabic and controlled Latin, and they became the major form of logic until the 20th century. The following table lists the names of the four types of propositions used in syllogisms and the corresponding sentence patterns that express them.

Type	Name	Pattern
<b>A</b>	<i>Universal affirmative</i>	Every $A$ is $B$ .
<b>I</b>	<i>Particular affirmative</i>	Some $A$ is $B$ .
<b>E</b>	<i>Universal negative</i>	No $A$ is $B$ .
<b>O</b>	<i>Particular negative</i>	Some $A$ is not $B$ .

With letters such as *A* and *B* in the sentence patterns, Aristotle introduced the first known use of variables in history. Each letter represents some category, which the Scholastics called *praedicatum* in Latin and which became *predicate* in English. If necessary, the verb form *is* may be replaced by *are*, *has*, or *have* in order to make grammatical English sentences. Although the patterns may look like English, they are limited to a highly constrained syntax and semantics: each sentence has exactly one quantifier, at most one negation, and a single predicate that is true or false of the individuals indicated by the subject.

Although Aristotle's syllogisms are the oldest version of formal logic, they are still an important subset of logic, which forms the foundation for description logics, such as DAML and OIL. For frame-like inheritance, the major premise is a universal affirmative statement with the connecting verb *is*; the minor premise is a universal or particular affirmative with *is*, *has*, or other verbs. Many constraints for a DBMS or an expert system can be stated as universal negative statements with any of the verbs. For constraint checking and constraint inheritance, the major premise is the constraint, and the minor premise is a statement in one of the other three patterns.

Another important version of logic is the Horn-clause subset, which is widely used for defining expert system rules and SQL views. The basic syntax has an if-then pattern: the if-part of the rule is a conjunction of one or more statements, which may have some negations; the then-part is a conjunction of one or more statements, which may not have negations. Following are two such rules for a library database, written in Attempto Controlled English (Fuchs et al. 1998; Schwitter 1998):

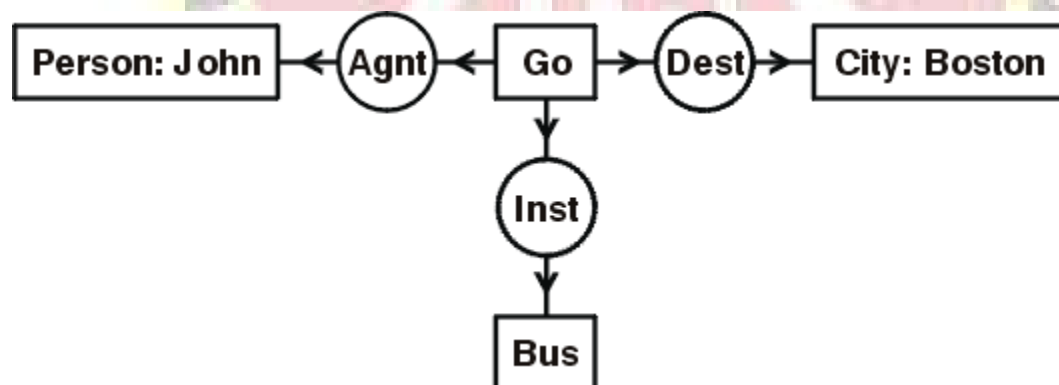
**If a copy of a book is checked out to a borrower  
and a staff member returns the copy  
then the copy is available.**

**If a staff member adds a copy of a book to the library  
and no catalog entry of the book exists  
then the staff member creates a catalog entry  
that contains the author name of the book  
and the title of the book  
and the subject area of the book**

**and the staff member enters the id of the copy  
and the copy is available.**

The Attempto system translates these rules to an executable form in Prolog. Anyone who can read English can read controlled English as if it were English, but controlled languages are formal languages that require some training for an author to stay within their limitations. Section 3 of this paper discusses tools that can guide an author to avoid the ambiguities of full natural language or help a human editor to clarify them.

In the late 19th century, three logically equivalent, but structurally very different notations for first-order logic (FOL) were developed. The first was the tree-like *Begriffsschrift* by Frege (1879), and the second was the algebraic notation by Peirce (1880, 1885). With minor modifications by Peano (1889), Peirce's version became the most commonly used notation for logic during the 20th century. The third notation was Peirce's *existential graphs* of 1897, which he called his *chef d'oeuvre*. KIF is a sorted version of Peirce's algebraic notation, and conceptual graphs are a sorted version of Peirce's graph notation. For comparison, Figure 1 is a CG representation of the controlled English sentence, **John is going to Boston by bus.**



**Figure 1: A conceptual graph in the display form**

The boxes in Figure 1 are called *concepts*, and the circles are called *conceptual relations*. The default quantifier for each concept is the existential, which says that something of the specified type exists; the concept [City: Boston] means that there exists a city, which is named Boston. Each conceptual relation has one or more *arcs*: (Agnt) links a concept that represents an action to the concept that represents its agent; (Inst) links the action to its instrument; and (Dest) links an

action that involves motion to its destination. All the relations in Figure 1 are *dyadic*, but in general, a conceptual relation may have any number of arcs.

Although the display form is quite readable, it is not easy to type or to transmit across a network. Therefore, two interchange formats have been developed: the Conceptual Graph Interchange Format (CGIF) maps directly to and from the display form; and the Knowledge Interchange Format (KIF) maps directly to and from the algebraic notation for predicate calculus. Following is the CGIF representation of Figure 1:

**[Go: \*x] [Person: 'John' \*y] [City: 'Boston' \*z] [Bus: \*w]**

**(Agnt ?x ?y) (Dest ?x ?z) (Inst ?x ?w)**

This statement captures every detail of the display form except the two-dimensional layout, which is not semantically relevant. If desired, the layout information could be included as structured comments inside the brackets and parentheses that enclose the nodes of the graph. The connections between concepts and relations, which are shown directly by the arcs of the graph in Figure 1, are shown indirectly by labels, such as ?x and ?y in CGIF. Those labels are translated to variables in KIF, as in the following example:

**(exists ((?x Go) (?y Person) (?z City) (?w Bus))**

**(and (Name ?y John) (Name ?z Boston)**

**(Agnt ?x ?y) (Dest ?x ?z) (Inst ?x ?w)))**

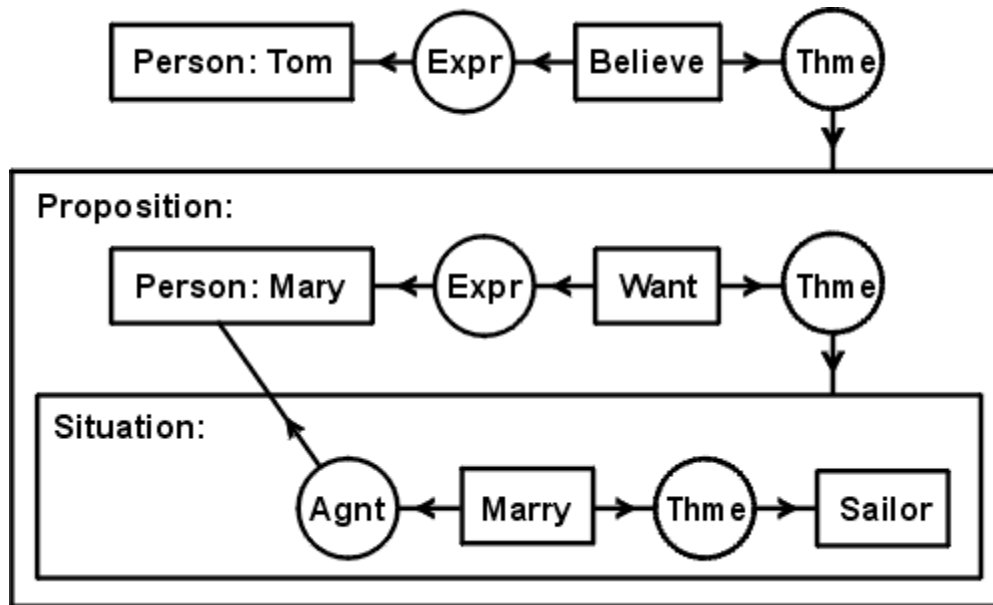
KIF notation is used for many theorem provers and inference engines that are based on predicate calculus. The translations between KIF and CGIF preserve the semantics: a mapping from KIF to CGIF and back to KIF might not generate an identical statement, but it will generate a statement that is logically equivalent.

KIF and conceptual graphs can represent the full range of operators and quantifiers of first-order logic, and they have been extended with metalevel features that can be used to define extensions to FOL, including modal logic and higher-order logic. The metalevel features are necessary for representing the speech acts of Elephant 2000, which uses logic to talk about the use of logic. In natural languages, metalevels are marked by a variety of syntactic features that delimit the context of the metalanguage from the context the object language. The most obvious delimiters are quotation marks, but similar contexts are introduced by verbs that express what some agent

says, thinks, believes, requests, wants, promises, or hopes. As an example, the following English sentence contains two nested levels, which are enclosed in brackets for emphasis:

**Tom believes [Mary wants [to marry a sailor]].**

This sentence is represented by the CG in Figure 2.



**Figure 2: A conceptual graph with two nested contexts**

The context of Tom's belief is represented by a concept of type Proposition, which contains a nested CG that states the proposition. The context of Mary's desire is represented by a concept of type Situation, which is described by a proposition that is stated by the nested CG. The (Expr) relation represents the experiencer of a mental state, and the (Thme) relation represents the theme. In general, the theme of a belief or an assertion is a proposition, but the theme of a desire must be something physical, such as a situation. Following is the CGIF equivalent of Figure 2:

```
[Person: *x1 'Tom'] [Believe *x2] (Expr ?x2 ?x1)
  (Thme ?x2 [Proposition:
    [Person: *x3 'Mary'] [Want *x4] (Expr ?x4 ?x3)
      (Thme ?x4 [Situation:
        [Marry *x5] (Agnt ?x5 ?x3) (Thme ?x5 [Sailor]) ] ] ) )
```

And following is the equivalent KIF statement.

```

(exists ((?x1 person) (?x2 believe))
  (and (name ?x1 'Tom) (expr ?x2 ?x1)
    (thme ?x2
      (exists ((?x3 person) (?x4 want) (?x8 situation))
        (and (name ?x3 'Mary) (expr ?x4 ?x3) (thme ?x4 ?x8)
          (dscr ?x8 (exists ((?x5 marry) (?x6 sailor))
            (and (Agnt ?x5 ?x3) (Thme ?x5 ?x6))))))))))

```

The context boxes delimit the scope of quantifiers and other logical operators. The sailor, whose existential quantifier occurs inside the context of Mary's desire, which itself is nested inside the context of Tom's belief, might not exist in reality. Following is another sentence that makes it clear that the sailor does exist:

**There is a sailor that Tom believes Mary wants to marry.**

This sentence corresponds to the CG in Figure 3.

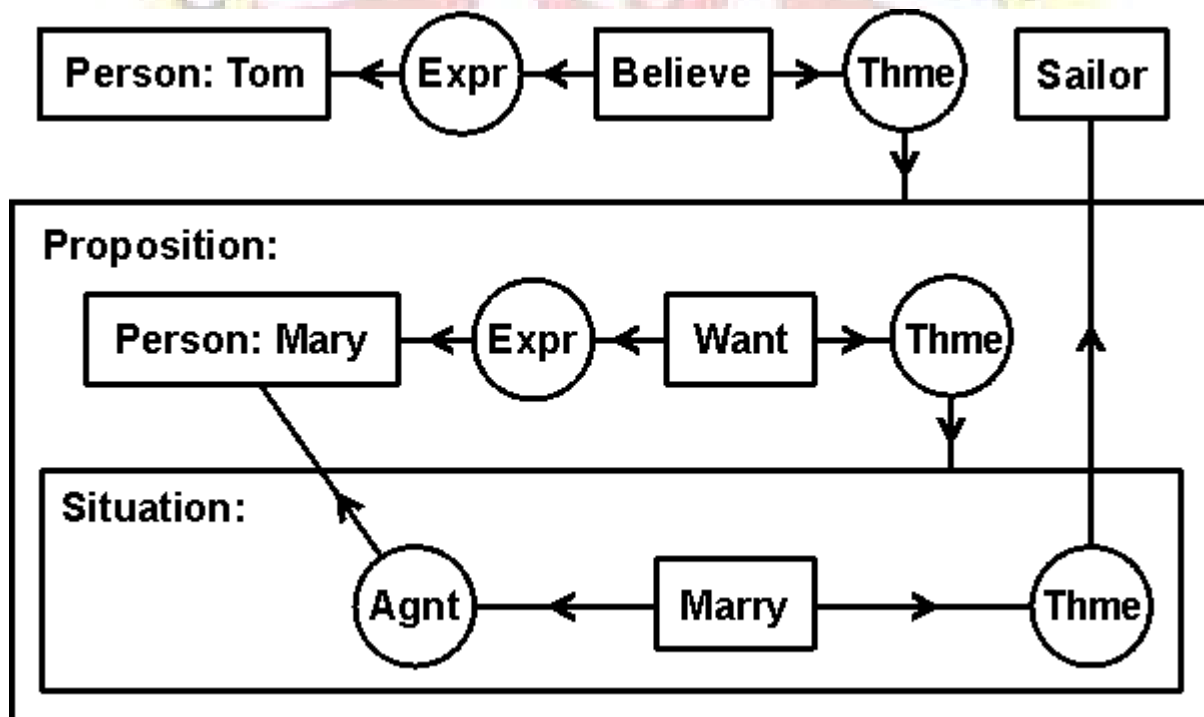


Figure 3: A CG that asserts the sailor's existence

The English sentence mentions the sailor before introducing any verb that creates a nested context. Therefore, the concept [Sailor] in Figure 3, with its implicit existential quantifier, is moved outside any nested context. In the CGIF and KIF notations, the concept or the quantifier that refers to the sailor would be moved to the front of the statement. Another possibility, represented by the sentence **Tom believes there is a sailor that Mary wants to marry**, could be represented by moving the concept [Sailor] into the middle context, which represents Tom's belief. In CGIF and KIF, the corresponding concept or quantifier would also be moved to the context of Tom's belief.

As these examples illustrate, conceptual graphs in the display form are more readable than either CGIF or KIF. There are two reasons for the improved readability:

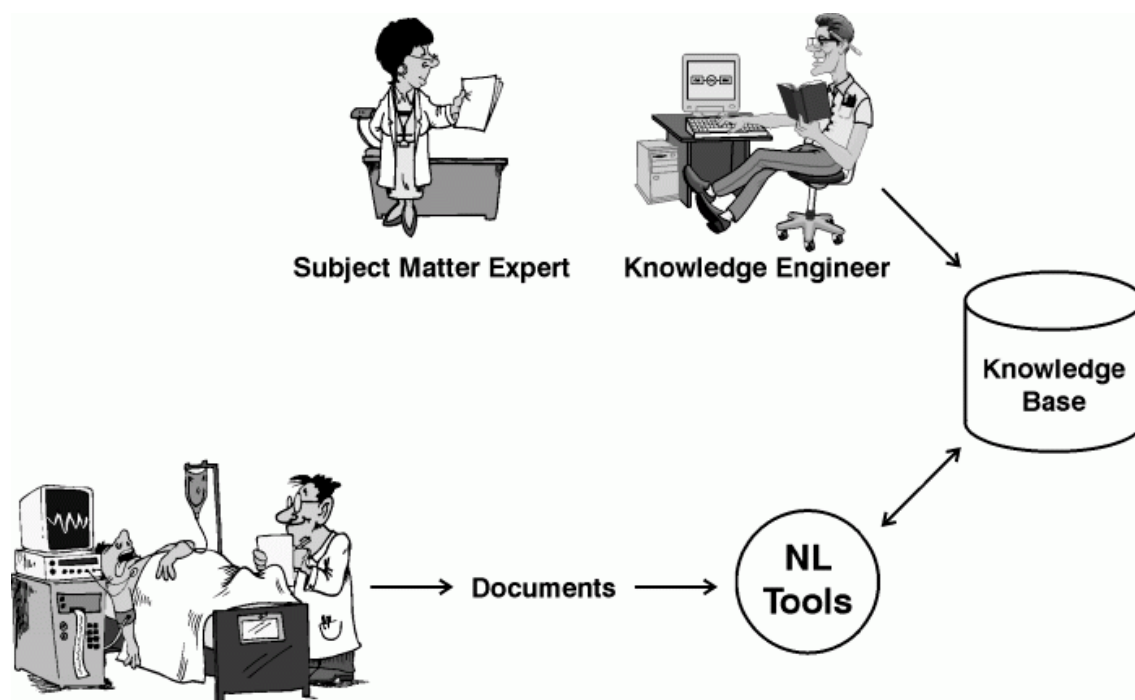
1. *Direct connections.* The arcs of the graph show connections directly without the need for labels or variables. In Figure 1, for example, the four concept boxes map to four distinct labels or variables in CGIF and KIF. To show the links to the relations, CGIF requires 10 occurrences of those labels, and KIF requires 12; furthermore, those occurrences are scattered throughout the linear strings.
2. *Nested enclosures.* As Figures 2 and 3 show, the contexts are shown more clearly with nested enclosures than with nested parentheses or brackets. By using both brackets and parentheses, CGIF has a slight advantage over KIF, but neither notation can compete with the nested boxes of the display form.

Besides human readability, graphs also have theoretical and computational advantages, which are discussed in Section 4.

### 3. Using Controlled Natural Languages

During the 1980s, the dominant approach to knowledge acquisition required two kinds of highly trained, highly paid professionals. At the top of Figure 4, a *knowledge engineer* is interviewing a *subject matter expert* in order to capture her knowledge and encode it in the arcane formats of an AI system. Meanwhile, computational linguists, who were designing natural-language tools, tried to make them translate NL documents into similar encodings without requiring any human intervention. At the bottom of Figure 4, a physician who is examining a patient scribbles some

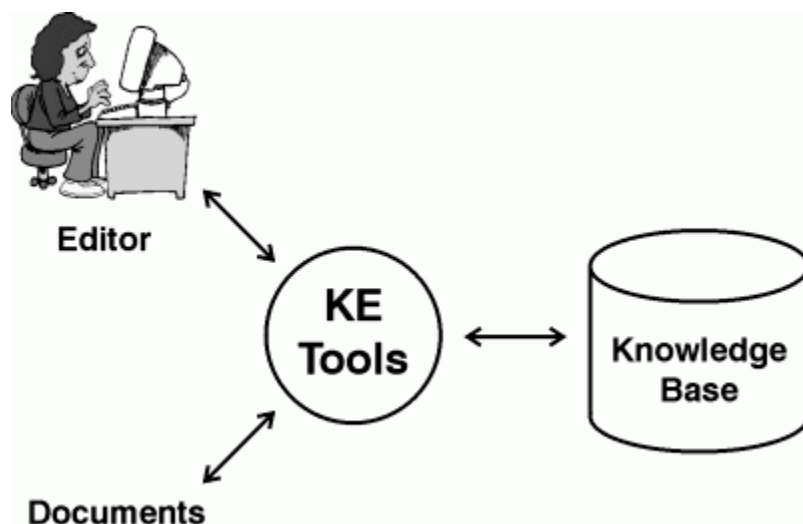
notes on a sheet of paper, which some clerk will later transcribe for the computer. Then the NL tools will attempt to convert those notes to the formats specified by the knowledge engineer.



**Figure 4: Twentieth century approaches to knowledge acquisition and NL processing**

There are two things wrong with Figure 4: the top row requires far too much human effort, and the bottom row is expected to process unrestricted natural language without any human assistance. To reduce the cost of two high-priced experts, some developers merged the two roles at the top row into one: either the subject matter expert learned knowledge engineering, or the knowledge engineer learned enough about the subject matter to extract knowledge from documents. Yet people with expertise in both fields became even more expensive to find, hire, and train. Figure 5 shows a better alternative: simplify the tools and the training required by the people who use them. Instead of designing complex NL tools that process documents without human intervention, AI researchers developed simpler *knowledge extraction* (KE) tools that can extract knowledge from documents with assistance from just one human editor. Furthermore, the editor communicates with the KE tools in a controlled natural language, which people can read without special training.





**Figure 5: Replacing two experts with one editor**

The editor in Figure 5 represents various people who at different times might play different roles with respect to the subject matter, the computer system, and the people and activities involved with them. Each of the three people in Figure 4 has a different kind of expertise. Any of them might use KE tools to edit their knowledge or to write a note, a report, or a book that someone else might edit with the aid of KE tools. Following are the three kinds of knowledge, the roles of the two experts in Figure 4, and the way that KE tools can help the editor in Figure 5 do the work of both:

- *Semantic knowledge.* The subject matter expert contributes the terminology and background knowledge that is typically recorded in textbooks, research reports, and reference manuals. That knowledge represents the semantics of the subject matter and its links to the natural language vocabulary. A editor in Figure 5 could use the KE tools to extract that knowledge from the documents, or the experts who write the documents could use the KE tools to generate a printable document and a knowledge base at the same time.
- *Episodic knowledge.* The physician at the patient's bedside contributes knowledge of particular instances or *episodes* in the day to day application of the subject matter. Instead of writing notes on a pad of paper, as in Figure 4, such people could enter that information into a computerized tablet or voice recognition system. The KE tools could

process the information immediately and respond with a paraphrase in a controlled natural language, which the operational personnel would correct and approve.

- *Patterns of language and logic.* The knowledge engineer in Figure 4 is a specialist in translating unformatted natural language to database tables, if-then rules, and procedural sequences. That kind of knowledge could be codified in a library of patterns or templates represented as conceptual graphs (Sowa 1999). The KE tools would apply the language patterns to extract information from documents and use the associated logic patterns to reformat it in a CNL. Since the KE process is not foolproof, a human editor must review, correct, and approve the output before it goes into the knowledge base.

From an editor's point of view, a KE system looks like an intelligent word processor combined with sophisticated tools for searching, classifying, summarizing, and paraphrasing. After the output has been revised by an editor, who might be the original author of the documents, the result can be stored in a knowledge base or be written as an annotation to the documents.

As examples of KE tools, Doug Skuce (1995, 1998, 2000) has designed an evolving series of knowledge extraction systems, which he called CODE, IKARUS, and DocKMan (Document-based Knowledge Management). All the input to the knowledge base, whether generated by the KE tools or entered directly by an editor, is represented in a CNL called ClearTalk. The KE tools have the following advantages over the older systems represented by Figure 4:

- *Reduced training for people.* A controlled natural language is a subset of the corresponding natural language. Anyone who can read English can immediately read ClearTalk, and the knowledge editors who write ClearTalk can learn to write it in a few hours. The ClearTalk system itself does most of the training through use: the restrictions are shown by menus and templates and enforced by immediate syntactic checks. By consistently using ClearTalk for all its output, the system reinforces the acceptable syntactic forms.
- *Reduced complexity in the system.* During the knowledge extraction process, the KE tools can ask the editor to resolve ambiguities in the documents, to select relevant passages, and to correct misinterpretations. After the knowledge has been translated to ClearTalk with human assistance, the restricted syntax of ClearTalk eliminates the

syntactic ambiguities of ordinary language. The semantic ambiguities are eliminated by the system, which enforces a single definition for every term. As a result, the system can automatically translate ClearTalk to and from logic and various computational languages.

- *Self-documenting systems.* People can read ClearTalk without special training, and a computer system can translate it automatically to a notation for logic, such as CGs or KIF. As a result, the comments and the implementation become identical, and there is never a discrepancy between what the human reads and what the machine is processing.
- *Document annotations.* The double-headed arrow in Figure 5 indicates that the ClearTalk output from the KE tools can also be written as an annotation to the original source documents. Those annotations can serve as humanly readable comments or as input to other ClearTalk systems.

As an example, the students in Skuce's operating systems course used the KE tools to map information from on-line Linux manuals to a knowledge base for a Linux help facility. The people who wrote the manuals were experts, but the students who edited the knowledge base were novice users of both Linux and the KE tools. As another example, Skuce built a simple knowledge base about animals for his 9-year-old daughter's school project. She and her class could browse the knowledge base on the web, and they had no difficulty in understanding every fact presented in ClearTalk.

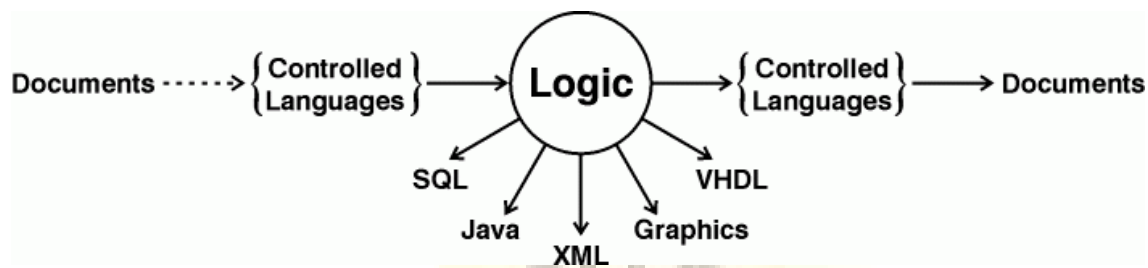
Over the past thirty years, many natural-language query systems have been developed that are much easier to use than SQL. Unfortunately, one major stumbling block has prevented them from becoming commercially successful: the amount of effort required to define the vocabulary terms and map them to the appropriate fields of the database is a large fraction of the effort required to design the database itself. However, if appropriate KE tools are used to design the database, the vocabulary needed for the query system can be generated as a by-product of the design process. As an example, the RÉCIT system (Rassinoux 1994; Rassinoux et al. 1998) uses KE tools to extract knowledge from medical documents written in English, French, or German and translates the results to a language-independent representation in conceptual graphs. The knowledge extraction process defines the appropriate vocabulary, specifies the database design, and adds new information to the database. The vocabulary generated by the KE process is sufficient for end users to ask questions and get answers in any of the three languages.

Translating an informal diagram to a formal notation of any kind is as difficult as translating informal English specifications to executable programs. But it is much easier to translate a formal representation in any version of logic to controlled natural languages, to various kinds of graphics, and to executable specifications. Walling Cyre and his students have developed KE tools for mapping both the text and the diagrams from patent applications and similar documents to conceptual graphs (Cyre et al. 1994, 1997, 1999). Then they implemented a scripting language for translating the CGs to circuit diagrams, block diagrams, and other graphic depictions. Their tools can also translate CGs to VHDL, a hardware design language used to specify *very high-speed integrated circuits* (VHSIC).

Design and specification languages have multiple metalevels. As an example, the Unified Modeling Language has four levels: the metametalanguage defines the syntax and semantics of the UML notations; the metalanguage defines the general-purpose UML types; a systems analyst defines application types as instances of the UML types; finally, the working data of an application program consists of instances of the application types. To provide a unified view of all these levels, Olivier Gerbé and his colleagues at the DMR Consulting Group implemented design tools that use conceptual graphs as the representation language at every level (Gerbé et al. 1995, 1996, 1997, 1998, 2000). For his PhD dissertation, Gerbé developed an ontology for using CGs as the metametalanguage for defining CGs themselves. He also applied it to other notations, including UML and the Common KADS system for designing expert systems. Using that theory, Gerbé and his colleagues developed the Method Repository System as an authoring environment for editing, storing, and displaying the methods used by the DMR consultants. Internally, the knowledge base is stored in conceptual graphs, but externally, the graphs can be translated to web pages in either English or French. About 200 business processes have been modeled in a total of 80,000 CGs. Since DMR is a Canadian company, the language-independent nature of CGs is important because it allows the specifications to be stored in the neutral CG form. Then any manager, systems analyst, or programmer can read them in his or her native language.

No single system discussed in this paper incorporates all the features desired in a KE system, but the critical research has been done, and the remaining work requires more development effort than pure research. Figure 6 shows the flow of information from documents to logic and then to documents or to various computational representations. The dotted arrow from documents to

controlled languages requires human assistance. The solid arrows represent fully automated translations that have been implemented in one or more systems.



**Figure 6: Flow of information from documents to computer representations**

For all these tools, the unifying representation language is logic, which could be represented in KIF, CGs, or other notations specialized for various tools. Aristotelian syllogisms together with Horn-clause rules provide sufficient expressive power to specify a Turing machine, and they support efficient computational mechanisms for executing the specifications. For database queries and constraints, statements in full first-order logic can be translated to SQL. All these subsets, however, use the same vocabulary of natural-language terms, which map to the same ontology of concepts and relations. From the user's point of view, the system communicates in a subset of natural language, and the differences between tools appear to be task-related differences rather than differences in language.

#### 4. Graph Algorithms

For many purposes, graphs are a natural representation that is isomorphic to the structure of an application: maps with cities as nodes and highways as arcs; flow diagrams through programs, electrical wiring, and plumbing; the valence bonds between atoms of an organic molecule; the communication links in a computer network; the reference patterns between documents and web sites on the Internet; and the semantics of natural languages with their complex phrase structures and anaphoric references. When such networks are represented by strings or matrices, the resulting data structures tend to make inefficient use of storage space, execution time, or both. This section surveys five important components of an intelligent system that can benefit from graph-based algorithms:

1. Storage, retrieval, and query.
2. Deductive reasoning for logical inference and theorem proving.
3. Inductive reasoning for learning new kinds of structures.
4. Abductive reasoning for discovering analogies.
5. Representing natural language semantics.

In all five of these areas, the direct connectivity of CGs and their nested contexts support algorithms that are simpler and more efficient than algorithms on linear strings and tables. For these reasons, most reasoning systems in AI, even those that use linear notations externally, use tree and graph data structures internally.

During the 1970s, the database field was embroiled in a controversy between the proponents of the new relational DBMS, which stored data in tables, and the proponents of older DBMS systems, which stored data in networks or hierarchies. For many applications, the network and hierarchical systems had better performance, but the relational systems became the universal standard because their logic-based query languages, such as SQL, were far easier to use than the navigational systems, which required a link-by-link traversal of the networks. The battle for network DBMS was finally lost when one of its staunchest defenders claimed that ease of use was not important because "programmers enjoy a challenge." Today, network systems have come back into vogue as the foundation for object-oriented DBMS, which represent the connections between objects more directly than the now standard RDBMS. Yet the query languages for OODBMS require the same kind of link-by-link traversals as the navigational methods of the 1970s. Unlike the logic-based SQL standard, the OODBMS query languages require far more programming effort, which must be specialized to the formats of each vendor.

To support a more natural interface between humans and computers, Sowa (1976, 1984) proposed conceptual graphs as an intermediate language between natural languages and logic-based computer languages. For question-answering systems, a CG derived from a natural language question could be translated to logic-based query languages such as SQL or be matched against the graphs of a network DBMS. In principle, CGs could provide a high-level interface for any DBMS — relational, network, or object-oriented. However, there were two obstacles to using CGs as the universal interface to every DBMS: the natural language processors were not

sufficiently robust to generate them, and the algorithms for generating answers from network databases were too slow.

The breakthrough in performance that made a CG database efficient was accomplished by Levinson and Ellis (1992), who developed algorithms that could search a lattice of graphs in logarithmic time. Instead of navigating the networks link by link, their systems could take any query graph  $q$  and determine where it fit within the lattice. As a result, it would return two pointers: one would point to the lower sublattice of all graphs that are more specialized than  $q$ , and the other would point to the upper sublattice of all graphs that are more generalized than  $q$ .

For deduction and theorem proving, Peirce (1897, 1909) discovered graph-based rules of inference, which are generalizations and simplifications of the rules of natural deduction by Gentzen (1935). The beauty of Peirce's rules is that they make a perfect fit with a system that stores and retrieves graphs in a generalization hierarchy: Peirce's rules are based on the conditions in which any graph  $p$  may be replaced by a generalization of  $p$  or a specialization of  $p$ . Furthermore, the negation of any context reverses the ordering for all graphs in the context: if  $p$  is a generalization of  $q$ , then  $\sim p$  is a specialization of  $\sim q$ . Esch and Levinson (1995, 1996) presented algorithms for combining Peirce's rules with search and retrieval from a generalization hierarchy, and one of Levinson's students, Stewart (1996), implemented those algorithms in a high-speed theorem prover for first-order logic. Every proposition that was proved, either as a theorem or as an intermediate result, was stored in its appropriate place in the generalization hierarchy together with a pointer to its proof. During a proof, each possible step that could be generated by Peirce's rules was used as a query graph  $q$  to determine whether  $q$  or any specialization of  $q$  had already been proved. If so, the proof was done.

A high-speed search and retrieval mechanism for generalization hierarchies of graphs can also be used as the basis for structural learning algorithms. Unlike neural networks and statistical algorithms, whose learning consists of changing numerical weights, a graph-based algorithm can learn arbitrarily large structures represented as graphs. To demonstrate that principle, Levinson (1996) used his search algorithms in a learning program that would learn to play board games, such as chess, by starting with no knowledge about the game other than the ability to make legal moves. His chess program, called Morph, learned chess by playing games with a tutor called

Gnu Chess, which was a master-level program, but it could not improve its performance by learning. At the end of each game, Morph was told whether the game was won, lost, or drawn (usually lost, especially in the early stages of learning). Then Morph would estimate the values of all the intermediate positions achieved during the game by backpropagation from the final value (1.0 for a win, 0.5 for a draw, or 0 for a loss), and save the chess positions with their estimated values as graphs in the hierarchy. When it made a move, Morph would determine all the possible moves, look up the corresponding positions in the hierarchy, find the closest matching positions, and consider their previously estimated values. Morph would then make the move that led to the position with the best estimated value. After playing a few thousand games with its tutor, Morph would have a sufficient database of moves with estimated values to play a decent game of chess.

To find analogies, Majumdar (2001) implemented a system called VivoMind, which represents knowledge in dynamic conceptual graphs. What makes the graphs dynamic are algorithms that pass messages along the nodes of a graph. Each node in a CG corresponds to an object that can pass messages to neighboring nodes. The result is an elegant generalization of the marker-passing algorithms originally implemented by Quillian (1966) and further developed by Fahlman (1979) and Hendler (1987). For finding analogies, VivoMind has proved to be as good or better than other analogical reasoners on all the usual test cases. It is also far more efficient computationally.

The contexts of conceptual graphs are based on Peirce's logic of existential graphs and his theory of indexicals. Yet the CG contexts happen to be isomorphic to the similarly nested *discourse representation structures* (DRS), which Hans Kamp (1981a,b) developed for representing and resolving indexicals in natural languages. When Kamp published his first version of DRS, he was not aware of Peirce's graphs. When Sowa (1984) published his book on conceptual graphs, he was not aware of Kamp's work. Yet the independently developed theories converged on semantically equivalent representations; therefore, Sowa and Way (1986) were able to apply Kamp's techniques to conceptual graphs. Such convergence is common in science; Peirce and Frege, for example, started from very different assumptions and converged on equivalent semantics for FOL, which 120 years later is still the most widely used version of logic.



Independently developed, but convergent theories that stand the test of time are a more reliable basis for standards than the consensus of a committee.

Although graphs are one of the most versatile representations, many good tools use other notations. A framework for intelligent systems should take advantage of different structural properties: some algorithms are more efficient on graphs, and some algorithms are more efficient on strings or tables. The logical equivalence of KIF and CGIF facilitates the mapping from one to the other. Their generality facilitates the integration with other languages that have more restricted expressive power, such as SQL, DAML, OIL, RDF, and others. Components based on any of those languages can be integrated with a system that uses KIF and CGs as its primary languages.

#### 5. Expressive Power and Computational Complexity

The limitations of AI systems have often been blamed on the complexity of the required computations. Various solutions have been proposed, ranging from highly parallel networks that mimic the mechanisms of the human brain to restricted languages that limit the complexity of the problem definition. A modular architecture could support components that use such strategies for special purposes: neural networks, for example, have been highly successful for pattern recognition, and restricted languages can be highly efficient for specific kinds of problems. For central communications among all components, however, the Elephant language used in the blackboard must be the most expressive, since it must transmit any information that any other component might use or generate. That extreme expressive power raises a question about the complexity of the computations needed to process it.

Computational complexity, however, is not a property of a language, but a property of the problems stated in that language. First-order logic has been criticized as computationally intractable because the proof of an arbitrary FOL theorem may take an exponentially increasing amount of time. That criticism, however, is misleading, since large numbers of problems stated in full FOL are easily solvable. Placing restrictions on the logic or the notation cannot make an intractable problem solvable; they merely make it impossible to state. The expressive power of Elephant does not slow down the communications from one component to another. The

components that receive a communication are responsible for determining what they can do with it.

For certain kinds of problems, first-order logic can be the most efficient way to express them and to solve them. A typical example is answering a query in terms of a relational database. The answer to an SQL query that uses the full expressive power of FOL can be evaluated in at most polynomial time, with the exponent of the polynomial equal to the number of quantifiers in the query. If the quantifiers range over an indexed domain, the evaluation can often be done in logarithmic time. Evaluating a constraint against a relational database is just as efficient as evaluating a query; in fact, every constraint can be translated to a corresponding query that asks for all instances in the database that violate the constraint. In commercial SQL systems, queries and constraints with the expressive power of FOL are routinely evaluated with databases containing gigabytes and terabytes of data.

Although the time to solve an intractable problem may be very long, the time to detect the complexity class of a problem can be very short. Callaghan (2001) took advantage of syntactic criteria to subdivide the Levinson-Ellis graph hierarchies into several disjoint subhierarchies, each of which is limited to one complexity class. For each subhierarchy, he determined appropriate algorithms for efficiently classifying and searching that hierarchy. To determine the complexity class of any graph, Callaghan computed a *signature* or descriptor to determine its complexity properties. Each graph's descriptor would specify easily computable prerequisites (necessary conditions) that any matching graph must meet. By precomputing the descriptor of a query graph, Callaghan accomplished several goals at once: determining the complexity of the search (tractability or decidability); narrowing the search to a particular class of graphs that have compatible descriptors; or determining whether the query graph lies outside the known complexity classes.

Besides the subhierarchies of graphs supported by the Levinson-Ellis algorithms, Callaghan's approach can accommodate any external subsystem for which a suitable descriptor can be computed by simple syntactic tests. Among those subsystems are the relational databases, which are highly optimized for data stored in tables. In fact, the Levinson-Ellis hierarchies are complementary to an RDBMS: the kinds of data that are most efficient with one are the least

efficient with the other. Other important subsystems include the specialized query languages of many versions of description logics. If a query graph lies outside of any of the known classes, it can be sent to a general first-order theorem prover. As a result, this approach can accept any query expressible in first-order logic, determine its complexity class, and send it to the most efficient subsystem for processing it.

Mapping a smaller logic to a more expressive logic is always possible, but the reverse mapping usually requires some restrictions. To map information from a large, rich knowledge base to a smaller, more efficiently computable one, Peterson, Andersen, and Engel (1998) developed a system they called the *knowledge bus*. Their source was the CYC knowledge base (Lenat & Guha 1990; Lenat 1995), which contains over 500,000 axioms expressed in full FOL with temporal, higher-order, and nonmonotonic extensions. Their target was a hybrid system that combined a relational database with an inference engine based on the Horn-clause subset of FOL. To map from one to the other, the knowledge bus performs the following transformations:

- *Extracting a subontology.* To extract an ontology for a particular application, the knowledge bus starts with a *seed* consisting of the concept types explicitly mentioned in the application. Then it searches through CYC to determine which axioms might deduce information about any of the seed types. Finally, it extracts those axioms together with the types and predicates used in them. For the sample application, it extracted approximately 1% of the total CYC knowledge base: 1531 types, 1267 predicates, and 5532 axioms.
- *Separating rules and constraints.* Since the Horn-clause inference engine cannot process arbitrary FOL statements, the knowledge bus separates the axioms into two classes: 4667 Horn-clause rules that are used for inferences, and 875 FOL statements that are used as database constraints. Both the inferencing and the constraint checking can be done efficiently, in at most polynomial time.
- *Restrictions and modifications.* For temporal reasoning, the knowledge bus adds extra arguments for starting and ending times to the CYC time-dependent predicates. To eliminate the higher-order features, it introduces constants of type Assertion. And to simulate the CYC nonmonotonic features, it uses a version of negation as failure.

For a particular application, the knowledge bus extracts a small subset of the CYC knowledge base that can be processed more efficiently by simpler tools. Although some information and some potential inferences are lost, the extracted subset has a well-founded semantics that is guaranteed to be free of contradictions. Furthermore, the resulting subset is more portable: the inference engine can be used as an extension to any relational database, and Engel (1999) has developed techniques for mapping the definitions and axioms to Java classes that can be used in web-based applications.

#### 6. A Flexible Modular Framework

A framework based on Elephant and Linda would subsume anything that could be done with a more conventional scripting language. Natural languages can specify procedures with a sequence of imperative statements linked by adverbs like *then* and *next*. A translation of those statements into KIF or CGs would specify the same procedure. But natural languages and their translations into logic could also specify more complex speech acts that could dynamically reconfigure the components of an intelligent system and their ways of interacting.

In the original Linda system, the operators access a blackboard that contains *tuples*, which consist of sequences of arbitrary data. For a system that supports multiple languages, the first element of the tuple should identify the language so that the Linda system could immediately determine how to interpret the remainder. A general format would have six elements:

1. *Language*. A character string that specifies the language, such as "KIF", "CGIF", "English", or "Deutsch".
2. *Source*. A character string that identifies the sender.
3. *Message Id*. A character string generated by the sender.
4. *Destination*. A character string that identifies the intended receiver, if known. For pattern-directed communications, this string is empty, and the message is matched to the patterns of available receivers.
5. *Speech Act*. A character string that states the speech act.
6. *Message*. An arbitrary expression in the specified language that states the propositional content of the message.

The Linda pattern matcher could use an ordinary string comparison for the first five elements of the tuple, but it would require a more general logical unification (of CGs or KIF statements) for the sixth. Unification of messages in controlled natural languages might be difficult to define, and the pattern matcher might need to translate the message to CGs or KIF if a pattern match is necessary.

To simulate a conventional scripting language, the destination would always be specified, and the speech act would always be "command". To access a relational database, the speech act would be "assertion" for an update, "question" for a query, or "definition" for creating a new table with a new format. At the end of his book, Austin (1962) specified a large number of possible speech acts, and he insisted that his list was not exhaustive. Following are his five categories, his description of each, and a few of his examples:

1. *Verdictives* "are typified by the giving of a verdict, as the name implies, by a jury, arbitrator, or umpire."

**Examples:** acquit, convict, calculate, estimate, measure, assess, characterize, diagnose.

2. *Exercitives* "are the exercising of powers, rights, or influence."

**Examples:** appoint, demote, excommunicate, command, direct, bequeath, claim, pardon, countermand, veto, dedicate.

3. *Commissives* "are typified by promising or otherwise undertaking."

**Examples:** promise, contract, undertake, intend, plan, propose, contemplate, engage, vow, consent, champion, oppose.

4. *Behabitives* "are a very miscellaneous group, and have to do with attitudes and *social behavior*."

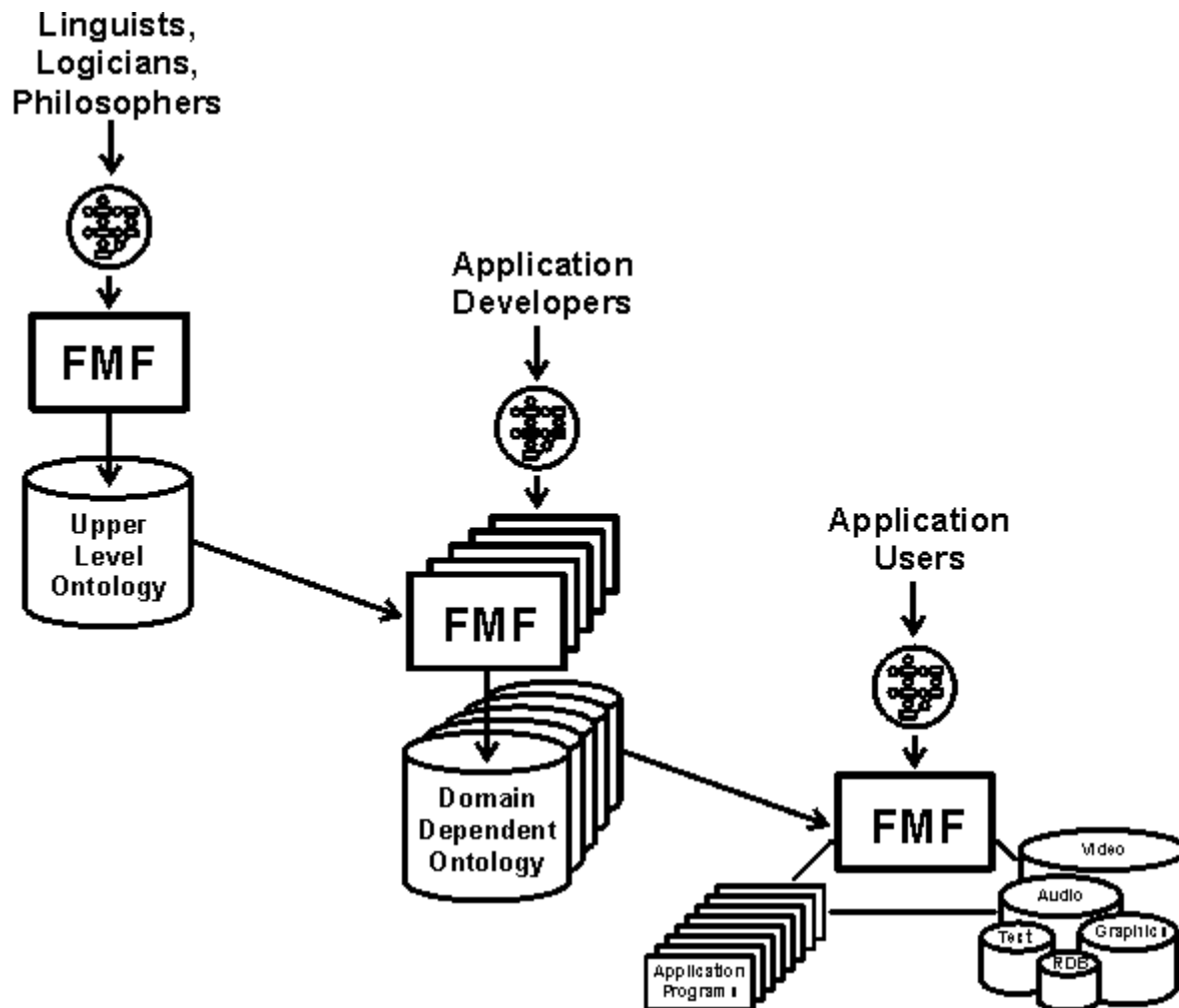
**Examples:** apologize, thank, deplore, congratulate, welcome, bless, curse, defy, challenge.

5. *Expositives* "make plain how our utterances fit into the course of an argument or conversation, how we are using words, or, in general, are expository."

**Examples:** affirm, deny, state, assert, ask, identify, remark, mention, inform, answer, repudiate, recognize, define, postulate, illustrate, explain, argue, correct, revise, tell, report, interpret.

The verbs listed in these examples illustrate the kinds of speech acts that people commonly perform, but in most cases, they omit the verb that specifies the speech act. A man who stands up in a meeting to shout something in an angry voice seldom begins with the words "I protest." Yet the people in the audience would recognize that the new speaker is protesting rather than agreeing with the previous speakers. Computers, however, need to be told how to interpret such speech, and an explicit statement of the speech act would enable them to respond more "intelligently."

To illustrate the kinds of speech acts in an AI system, Figure 7 shows a kind of system discussed by Sowa (2000). The boxes labeled FMF represent the same flexible modular framework that has been adapted to different kinds of tasks by changing the roles and the kinds of speech acts expected of the users. At the upper left, logicians, linguists, and philosophers are using the FMF to define a general ontology. Logicians could use FMF to enter the definitions and axioms for logical operators, set theory, and basic mathematical concepts and relations. Linguists could use it to enter the grammar rules of natural languages and the kinds of semantic types and relations. Philosophers could use FMF to collaborate with the linguists and logicians in analyzing and defining the fundamental ontologies of space, time, and causality common to all domains of application. The major speech act for these users would be definition, but they might also ask questions about how to use the system, and they might use verdictives to evaluate the work of their colleagues.



**Figure 7: Tailoring an FMF for different purposes**

In the center of Figure 7, application developers use FMF to enter domain-dependent information about specific applications. Some of them would use FMF to define generic ontologies for industries such as banking, agriculture, mining, education, and manufacturing. Others would start with one or more generic ontologies and combine them or tailor them to a particular business, project, or application. The users in this mode would perform the same kinds of speech acts as the logicians, linguists, and philosophers. But they might put more emphasis on commissives, which would commit them to strict deadlines and performance goals.

At the bottom right of Figure 7 the application users might interact with the FMF in an unpredictable number of ways. A business user with a job to do would have different requirements from a recreational user. Both, however, might react with behabitives, such as

grumbling, complaining, or cursing, when the system doesn't do what they wish. But unlike more conventional systems, an FMF could apologize, sympathize, and commiserate.

The examples shown in Figure 7 do not begin to exploit the kinds of opportunities offered by an FMF that is able to recognize and respond to a wide range of speech acts. An important reason for building an FMF is to explore new ways of interaction, either between computers and humans or among mixed committees of human and computer participants. The explicit recognition and marking of speech acts enables the components of an FMF to interact, negotiate, and cooperate more intelligently among themselves and with their human users.

## 7. Implementing the FMF

A major advantage of a flexible modular framework is that it doesn't have to be implemented all at once. The four design principles, which enabled Unix-like systems to be implemented on anything from a wearable computer to the largest supercomputers, can also support the growth of intelligent systems from simple beginnings to a large "society of mind," as Minsky (1985) called it. For an initial implementation, each of the four principles could be reduced to the barest minimum, but any of them could be enhanced incrementally without disturbing any previously supported operations:

1. The first component that must be implemented is a blackboard for passing messages. Even the pattern matcher might be omitted in the first implementation, and messages could only be sent to named destinations. For greater power and flexibility, a pattern matcher is necessary, but the basic Linda systems use only a simple pattern matcher that is far less sophisticated than most AI systems. The greatest power would come from patterns stored in a hierarchy of graphs based on the Levinson-Ellis algorithms, which could accommodate millions of patterns that might invoke intelligent agents distributed anywhere across the Internet.
2. Any components that accept inputs and generate outputs could be accommodated in an FMF. The first implementation might support only conventional components that do exactly what they are told, such as the traditional collection of Unix utilities. More sophisticated components for reasoning, planning, problem solving, and natural language



processing could be added incrementally. The initial implementation would look like a pattern-matching front-end to a Unix command line, but it could grow arbitrarily far. Each stage in the growth would continue to have the full functionality of every preceding stage. Nothing would become obsolete as more intelligence is added by the new components.

3. The Elephant language, which serves as the glue for linking components, could also grow incrementally. An initial version could consist of just the three verbs implemented in conventional computer systems: *tell*, *ask*, and *do*. Those verbs correspond to the declarative, interrogative, and imperative moods of English. They also correspond to the three operators supported by a Linda blackboard: *output*, *input*, and *execute*. Those verbs would be necessary to support many, if not most of the messages in even the most sophisticated systems. As more sophisticated components are added to the FMF, other verbs could be added to support more complex interactions: *authorize* for secure communications; *reply*, *lock*, and *commit* for transactions that require multiple exchanges; *explain* for help facilities; and *promise* for future commitments.
4. The communication language for writing messages could be conventional first-order logic, which supports a wide range of simpler subsets, such as lists, frames and rules. The richer CG language includes FOL, but with nested contexts and metalevel statements about the contents of any context. New languages and dialects of languages could be added whenever a translator becomes available for mapping them to and from the patterns of CGs that are used by the blackboard communication center.

As an FMF is being developed, it can accommodate any mixture of a variety of components: newly designed components specially tailored for the FMF; legacy systems enclosed in a wrapper that translates their I/O formats to the common language of the blackboard; commercial products that perform specific services; experimental components that are being designed and tested in research projects; an open-ended variety of client interfaces specialized for different applications; remote servers distributed anywhere across the Internet.

A blackboard is an ideal platform for supporting hot-swap or plug-n-play components. When a new component is added to the FMF, it would send a message to the blackboard to identify itself and the patterns of messages it accepts. It could then be invoked by any other component whose

message matches the appropriate pattern. To take advantage of that flexibility, the Jini system uses the Linda operators to accommodate any kind of I/O device that might be attached to a network. But for intelligent systems, it is even more important to have that flexibility at the center instead of the periphery.

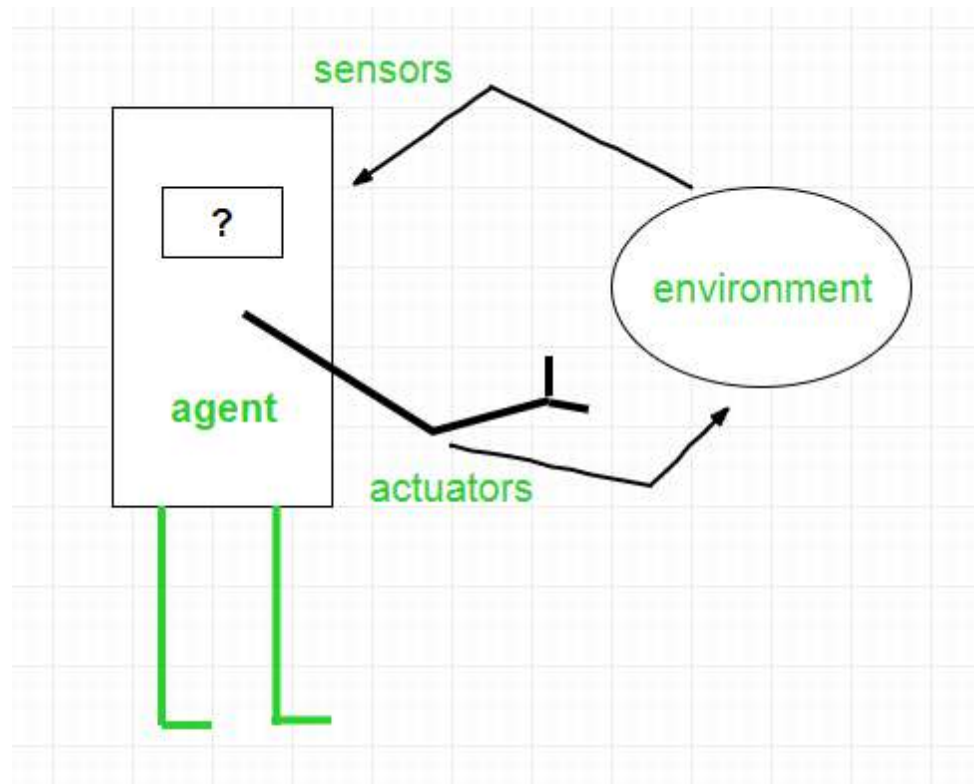
Any server anywhere on the Internet could be converted to an intelligent agent by using an FMF as its front end. It could then respond to requests from other FMF servers anywhere else on the Internet. Each FMF would be, in Minsky's terms, a society of mind, and the entire Internet would become a society of societies. Human users could have a personal FMF running on their own computers, which could communicate with any other FMF to request services. The traditional help desks, in which a human expert answers the same questions repeatedly for multiple users, could be replaced by a human teacher or editor, as in Figure 5, who would build a knowledge base. That knowledge base would drive a specialized FMF, which could be consulted by the personal FMF of anyone who asks a relevant question. The intelligence accessible to any user would then be the combined intelligence of his or her personal FMF together with every FMF accessible to it across the Internet.

### **LOGICAL AGENTS**

Artificial intelligence is defined as a study of rational agents. A rational agent could be anything which makes decisions, as a person, firm, machine, or software. It carries out an action with the best outcome after considering past and current percepts(agent's perceptual inputs at a given instance). An AI system is composed of an **agent and its environment**. The agents act in their environment. The environment may contain other agents. An agent is anything that can be viewed as :

- perceiving its environment through **sensors** and
- acting upon that environment through **actuators**

**Note :** Every agent can perceive its own actions (but not always the effects)

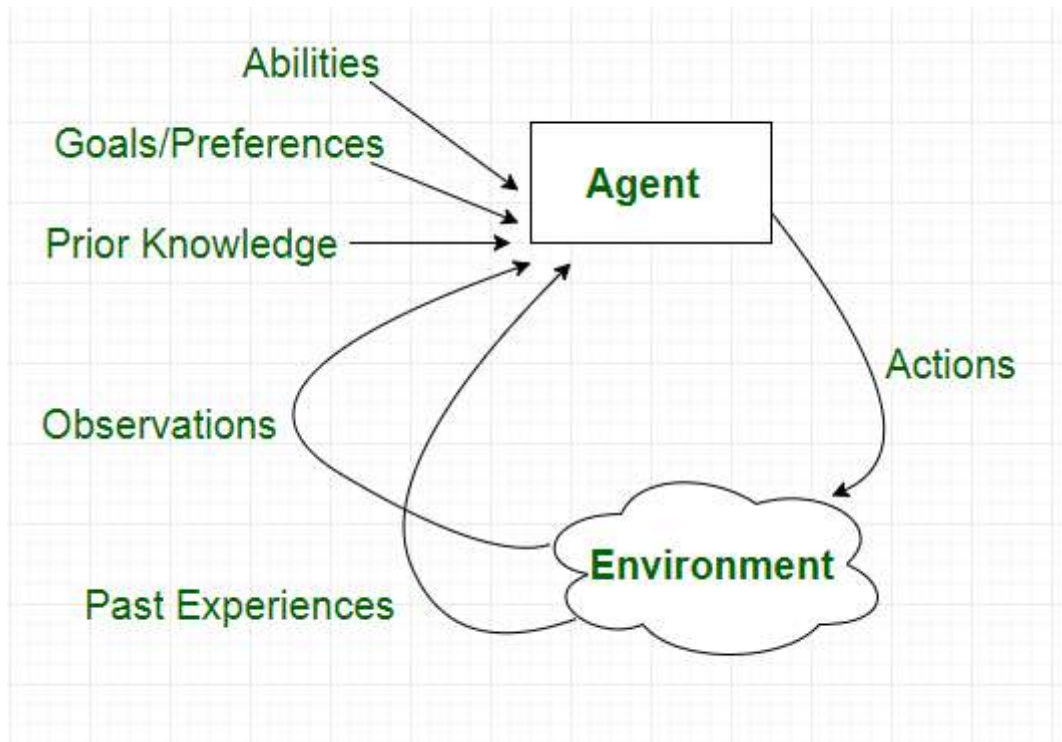


To understand the structure of Intelligent Agents, we should be familiar with *Architecture* and *Agent Program*. **Architecture** is the machinery that the agent executes on. It is a device with sensors and actuators, for example : a robotic car, a camera, a PC. **Agent program** is an implementation of an agent function. An **agent function** is a map from the percept sequence (history of all that an agent has perceived till date) to an action.

*Agent = Architecture + Agent Program*

Examples of Agent:-  
 A **software agent** has Keystrokes, file contents, received network packages which act as sensors and displays on the screen, files, sent network packets acting as actuators.  
 A **Human agent** has eyes, ears, and other organs which act as sensors and hands, legs, mouth, and other body parts acting as actuators.  
 A **Robotic agent** has Cameras and infrared range finders which act as sensors and various

motors acting as actuators.



### Types of Agents

Agents can be grouped into four classes based on their degree of perceived intelligence and capability :



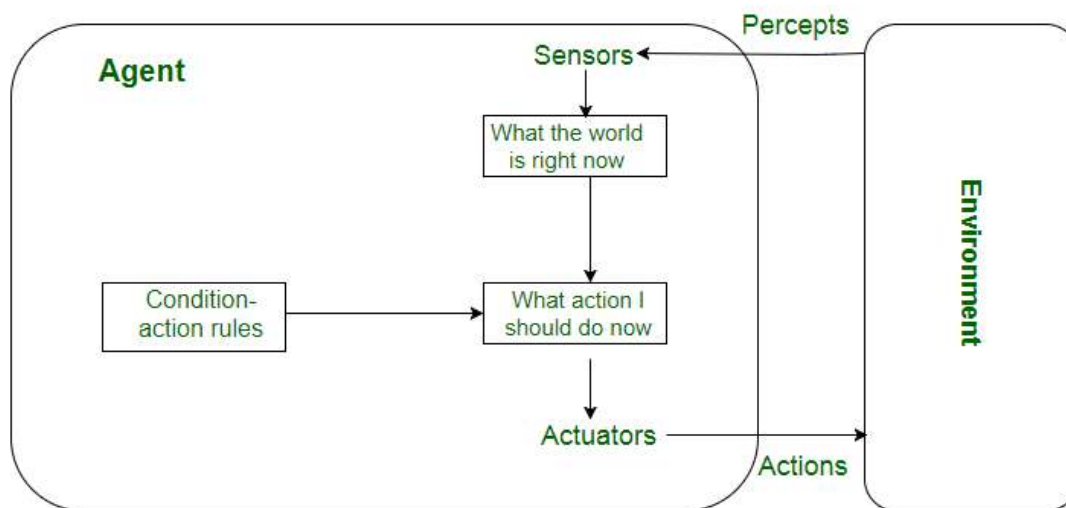
- Simple Reflex Agents
- Model-Based Reflex Agents
- Goal-Based Agents
- Utility-Based Agents
- Learning Agent

### Simple reflex agents

Simple reflex agents ignore the rest of the percept history and act only on the basis of the **current percept**. Percept history is the history of all that an agent has perceived till date.

The agent function is based on the **condition-action rule**. A condition-action rule is a rule that maps a state i.e, condition to an action. If the condition is true, then the action is taken, else not. This agent function only succeeds when the environment is fully observable. For simple reflex agents operating in partially observable environments, infinite loops are often unavoidable. It may be possible to escape from infinite loops if the agent can randomize its actions. Problems with Simple reflex agents are :

- Very limited intelligence.
- No knowledge of non-perceptual parts of state.
- Usually too big to generate and store.
- If there occurs any change in the environment, then the collection of rules need to be updated.

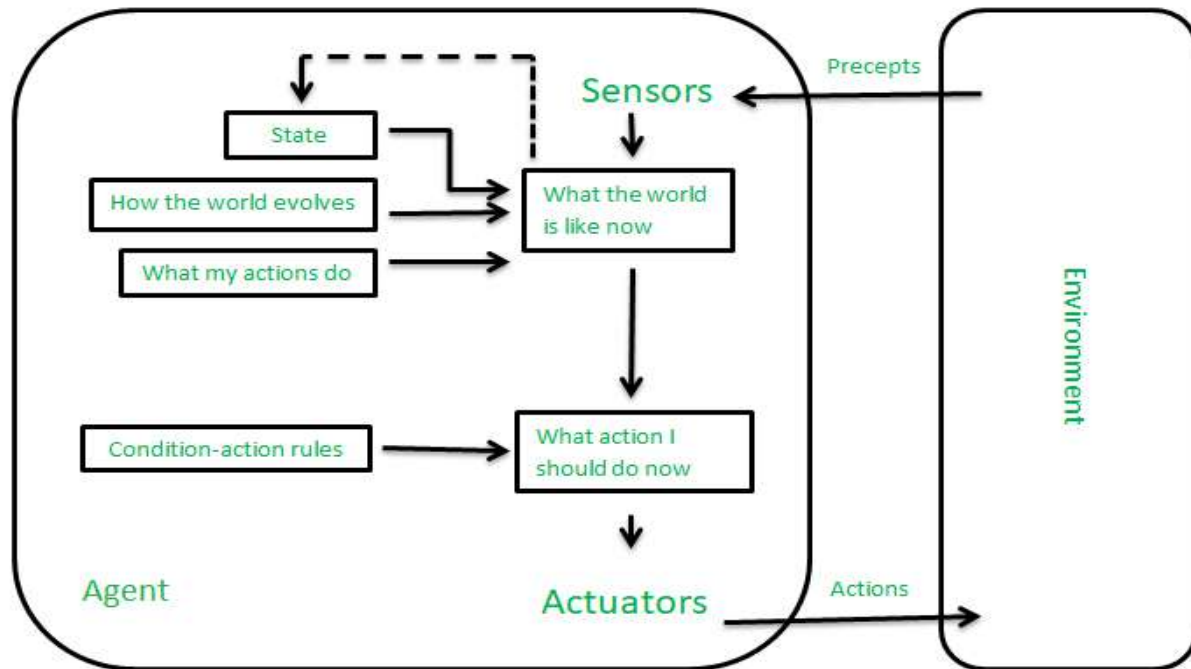


### Model-based reflex agents

It works by finding a rule whose condition matches the current situation. A model-based agent can handle **partially observable environments** by use of model about the world. The agent has to keep track of **internal state** which is adjusted by each percept and that depends on the percept history. The current state is stored inside the agent which maintains some kind of structure describing the part of the world which cannot be seen. Updating the state requires information about :

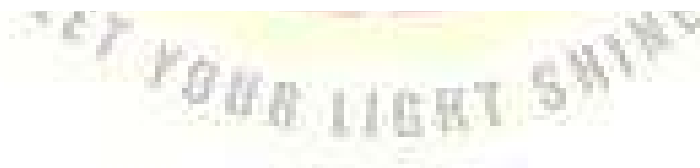
- how the world evolves in-dependently from the agent, and

- how the agent actions affects the world.

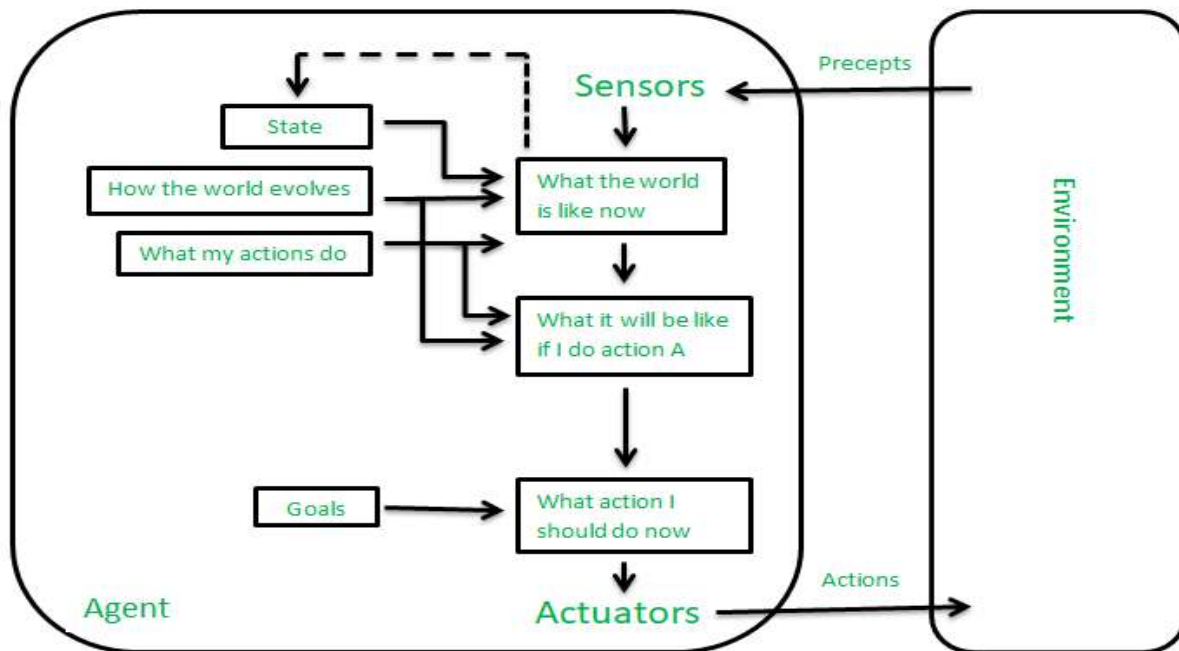


### Goal-based agents

These kind of agents take decision based on how far they are currently from their **goal**(description of desirable situations). Their every action is intended to reduce its distance from the goal. This allows the agent a way to choose among multiple possibilities, selecting the one which reaches a goal state. The knowledge that supports its decisions is represented explicitly and can be modified, which makes these agents more flexible. They



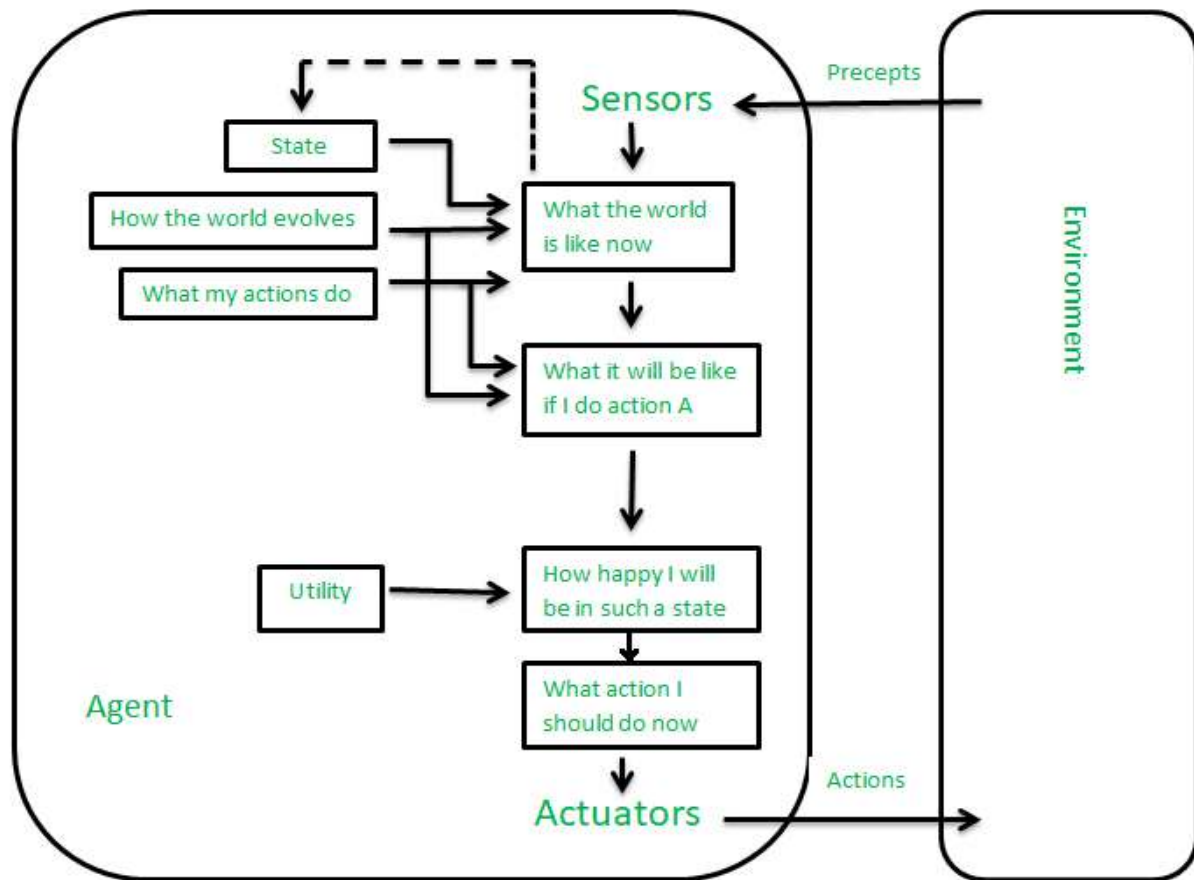
usually require search and planning. The goal-based agent's behavior can easily be changed.



### Utility-based agents

The agents which are developed having their end uses as building blocks are called utility based agents. When there are multiple possible alternatives, then to decide which one is best, utility-based agents are used. They choose actions based on a **preference (utility)** for each state. Sometimes achieving the desired goal is not enough. We may look for a quicker, safer, cheaper trip to reach a destination. Agent happiness should be taken into consideration. Utility describes how **“happy”** the agent is. Because of the uncertainty in the world, a utility agent chooses the action that maximizes the expected utility. A utility function maps a state onto a

real number which describes the associated degree of happiness.



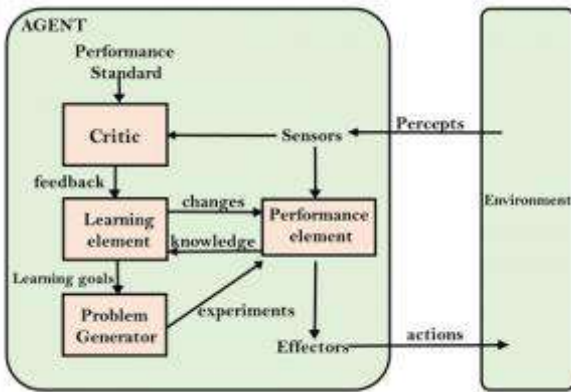
### Learning Agent

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:

1. **Learning element** :It is responsible for making improvements by learning from the environment
2. **Critic**: Learning element takes feedback from critic which describes how well the agent is doing with respect to a fixed performance standard.
3. **Performance element**: It is responsible for selecting external action
4. **Problem Generator**: This component is responsible for suggesting actions that will lead to new and informative experiences.





### 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:
  - a. **Syntax**
  - b. **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 shorthand notation in FOL.

### Basic Elements of First-order logic:

Following are the basic elements of FOL syntax:

<b>Constant</b>	1, 2, A, John, Mumbai, cat,.....
<b>Variables</b>	x, y, z, a, b,....
<b>Predicates</b>	Brother, Father, >,....
<b>Function</b>	sqrt, LeftLegOf, ....
<b>Connectives</b>	$\wedge, \vee, \neg, \Rightarrow, \Leftrightarrow$

<b>Equality</b>	$=$
<b>Quantifier</b>	$\forall, \exists$

### 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:  $\Rightarrow$  Brothers(Ravi, Ajay).  
Chinky is a cat:  $\Rightarrow$  cat (Chinky).

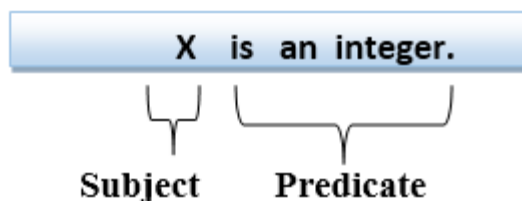
### Complex Sentences:

- 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  $\forall$ , which resembles an inverted A.

**Note:** *In universal quantifier we use implication " $\rightarrow$ ".*

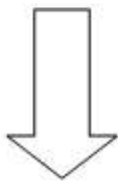
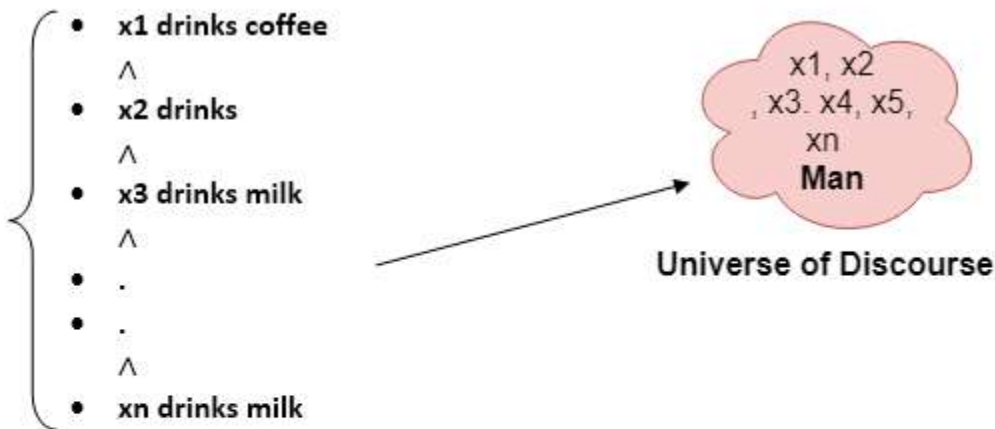
If x is a variable, then  $\forall x$  is read as:

- **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:



So in shorthand notation, we can write it as :

$\forall x \text{ man}(x) \rightarrow \text{drink}(x, \text{coffee}).$

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  $\exists$ , 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 ( $\wedge$ ).**

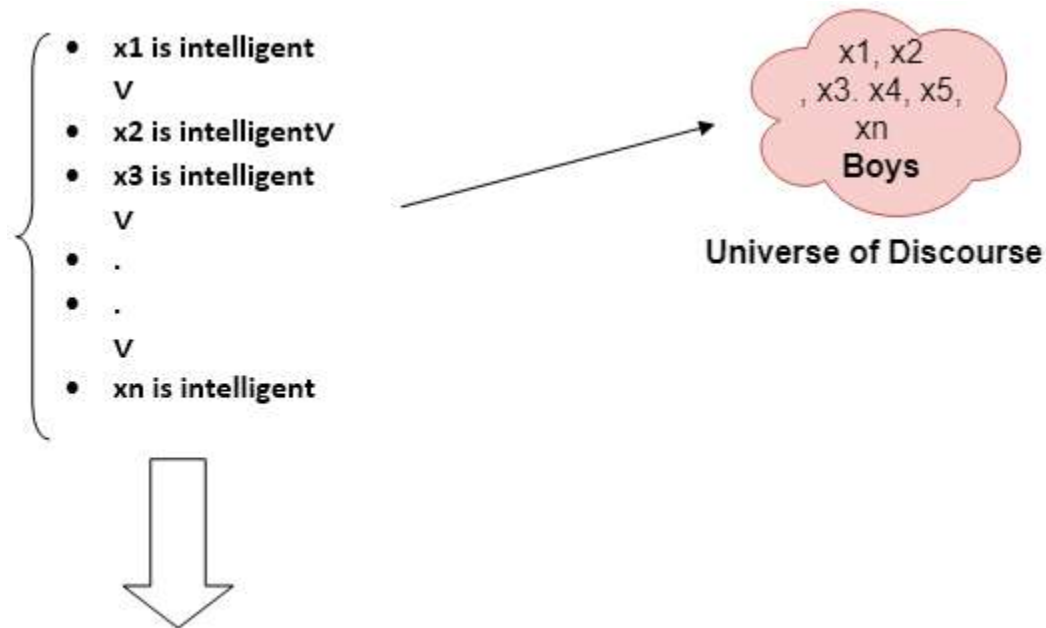
If x is a variable, then existential quantifier will be  $\exists x$  or  $\exists(x)$ . And it will be read as:

- **There exists a 'x.'**
- **For some 'x.'**

- For at least one 'x.'

### Example:

Some boys are intelligent.



So in short-hand notation, we can write it as:

**Ex: boys(x)  $\wedge$  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  $\forall$  is implication  $\rightarrow$ .
- The main connective for existential quantifier  $\exists$  is and  $\wedge$ .

### Properties of Quantifiers:

- In universal quantifier,  $\forall x\forall y$  is similar to  $\forall y\forall x$ .
- In Existential quantifier,  $\exists x\exists y$  is similar to  $\exists y\exists x$ .
- $\exists x\forall y$  is not similar to  $\forall y\exists x$ .

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.

$$\forall x \text{ bird}(x) \rightarrow \text{fly}(x).$$

**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  $\forall$ , and it will be represented as follows:

$$\forall x \text{ man}(x) \rightarrow \text{respects}(x, \text{parent}).$$

**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  $\exists$ , and it will be represented as:

$$\exists x \text{ boys}(x) \rightarrow \text{play}(x, \text{cricket}).$$

**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  $\forall$  with negation, so following representation for this:

$$\neg \forall (x) [ \text{student}(x) \rightarrow \text{like}(x, \text{Mathematics}) \wedge \text{like}(x, \text{Science}) ].$$

**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:

$$\exists (x) [ \text{student}(x) \rightarrow \text{failed}(x, \text{Mathematics}) \wedge \forall (y) [ \neg(x=y) \wedge \text{student}(y) \rightarrow \neg \text{failed}(y, \text{Mathematics}) ] ].$$

### 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:**  $\forall x \exists (y)[P(x, y, z)]$ , 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:**  $\forall x [A(x) B(y)]$ , here  $x$  and  $y$  are the bound variables.

### Knowledge Engineering in First-order logic

#### What is knowledge-engineering?

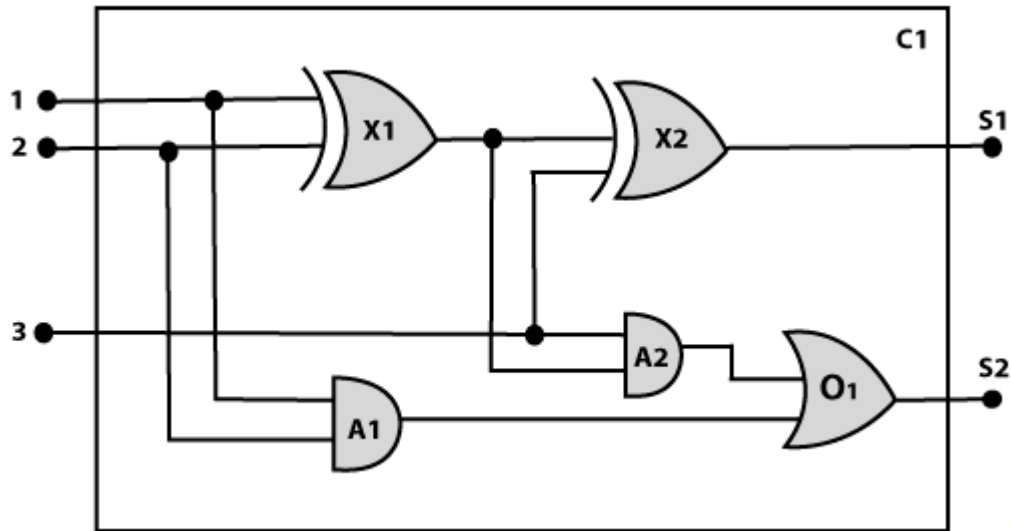
The process of constructing a knowledge-base in first-order logic is called as knowledge-engineering. In **knowledge-engineering**, someone who investigates a particular domain, learns important concept of that domain, and generates a formal representation of the objects, is known as **knowledge engineer**.

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.  $\forall t1, t2 \text{ Terminal}(t1) \wedge \text{Terminal}(t2) \wedge \text{Connect}(t1, t2) \rightarrow \text{Signal}(t1) = \text{Signal}(t2)$ .

- Signal at every terminal will have either value 0 or 1, it will be represented as:

1.  $\forall t \text{ Terminal}(t) \rightarrow \text{Signal}(t) = 1 \vee \text{Signal}(t) = 0.$

- Connect predicates are commutative:

1.  $\forall t_1, t_2 \text{ Connect}(t_1, t_2) \rightarrow \text{Connect}(t_2, t_1).$

- Representation of types of gates:

1.  $\forall g \text{ Gate}(g) \wedge r = \text{Type}(g) \rightarrow r = \text{OR} \vee r = \text{AND} \vee r = \text{XOR} \vee r = \text{NOT}.$

- Output of AND gate will be zero if and only if any of its input is zero.

1.  $\forall g \text{ Gate}(g) \wedge \text{Type}(g) = \text{AND} \rightarrow \text{Signal}(\text{Out}(1, g)) = 0 \Leftrightarrow \exists n \text{ Signal}(\text{In}(n, g)) = 0.$

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

1.  $\forall g \text{ Gate}(g) \wedge \text{Type}(g) = \text{OR} \rightarrow \text{Signal}(\text{Out}(1, g)) = 1 \Leftrightarrow \exists n \text{ Signal}(\text{In}(n, g)) = 1$

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

1.  $\forall g \text{ Gate}(g) \wedge \text{Type}(g) = \text{XOR} \rightarrow \text{Signal}(\text{Out}(1, g)) = 1 \Leftrightarrow \text{Signal}(\text{In}(1, g)) \neq \text{Signal}(\text{In}(2, g)).$

- Output of NOT gate is invert of its input:

1.  $\forall g \text{ Gate}(g) \wedge \text{Type}(g) = \text{NOT} \rightarrow \text{Signal}(\text{In}(1, g)) \neq \text{Signal}(\text{Out}(1, g)).$

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

1.  $\forall g \text{ Gate}(g) \wedge \text{Type}(g) = \text{NOT} \rightarrow \text{Arity}(g, 1, 1)$

2.  $\forall g \text{ Gate}(g) \wedge r = \text{Type}(g) \wedge (r = \text{AND} \vee r = \text{OR} \vee r = \text{XOR}) \rightarrow \text{Arity}(g, 2, 1).$

- All gates are logic circuits:

1.  $\forall g \text{ Gate}(g) \rightarrow \text{Circuit}(g).$

### 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 atomic 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:  $\text{Type}(x1) = \text{XOR}, \text{Type}(X2) = \text{XOR}$
2. For AND gate:  $\text{Type}(A1) = \text{AND}, \text{Type}(A2) = \text{AND}$
3. For OR gate:  $\text{Type}(O1) = \text{OR}$ .

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.  $\exists i1, i2, i3 \text{ Signal}(\text{In}(1, C1))=i1 \wedge \text{Signal}(\text{In}(2, C1))=i2 \wedge \text{Signal}(\text{In}(3, C1))=i3$
2.  $\wedge \text{Signal}(\text{Out}(1, C1))=0 \wedge \text{Signal}(\text{Out}(2, C1))=1$

### 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  $1 \neq 0$ .

### What is knowledge representation?

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

- Knowledge representation and reasoning (KR, KRR) is the part of Artificial intelligence which concerned with AI agents thinking and how thinking contributes to intelligent behavior of agents.
- It is responsible for representing information about the real world so that a computer can understand and can utilize this knowledge to solve the complex real world problems such as diagnosis a medical condition or communicating with humans in natural language.
- It is also a way which describes how we can represent knowledge in artificial intelligence. Knowledge representation is not just storing data into some database, but it also enables an intelligent machine to learn from that knowledge and experiences so that it can behave intelligently like a human.

### What to Represent:

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

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

(Here, sentences are used as a technical term and not identical with the English language).

**Knowledge:** Knowledge is awareness or familiarity gained by experiences of facts, data, and situations. Following are the types of knowledge in artificial intelligence:

### Types of knowledge

Following are the various types of knowledge:



#### 1. Declarative Knowledge:

- Declarative knowledge is to know about something.
- It includes concepts, facts, and objects.
- It is also called descriptive knowledge and expressed in declarative sentences.
- It is simpler than procedural language.

## 2. Procedural Knowledge

- It is also known as imperative knowledge.
- Procedural knowledge is a type of knowledge which is responsible for knowing how to do something.
- It can be directly applied to any task.
- It includes rules, strategies, procedures, agendas, etc.
- Procedural knowledge depends on the task on which it can be applied.

## 3. Meta-knowledge:

- Knowledge about the other types of knowledge is called Meta-knowledge.

## 4. Heuristic knowledge:

- Heuristic knowledge is representing knowledge of some experts in a field or subject.
- Heuristic knowledge is rules of thumb based on previous experiences, awareness of approaches, and which are good to work but not guaranteed.

## 5. Structural knowledge:

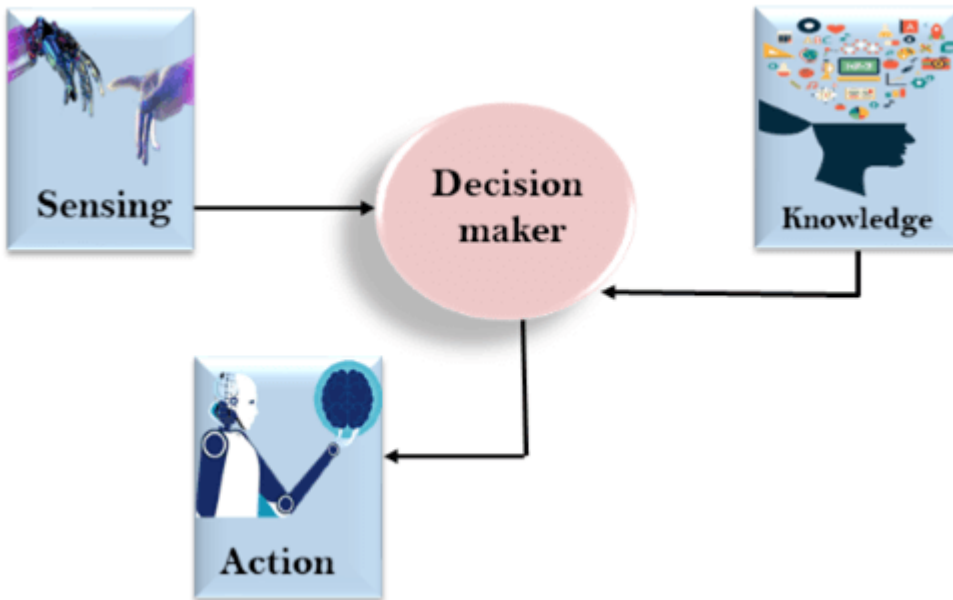
- Structural knowledge is basic knowledge to problem-solving.
- It describes relationships between various concepts such as kind of, part of, and grouping of something.
- It describes the relationship that exists between concepts or objects.

## The relation between knowledge and intelligence:

Knowledge of real-worlds plays a vital role in intelligence and same for creating artificial intelligence. Knowledge plays an important role in demonstrating intelligent behavior in AI agents. An agent is only able to accurately act on some input when he has some knowledge or experience about that input.

Let's suppose if you met some person who is speaking in a language which you don't know, then how you will be able to act on that. The same thing applies to the intelligent behavior of the agents.

As we can see in below diagram, there is one decision maker which acts by sensing the environment and using knowledge. But if the knowledge part will not be present then, it cannot display intelligent behavior.

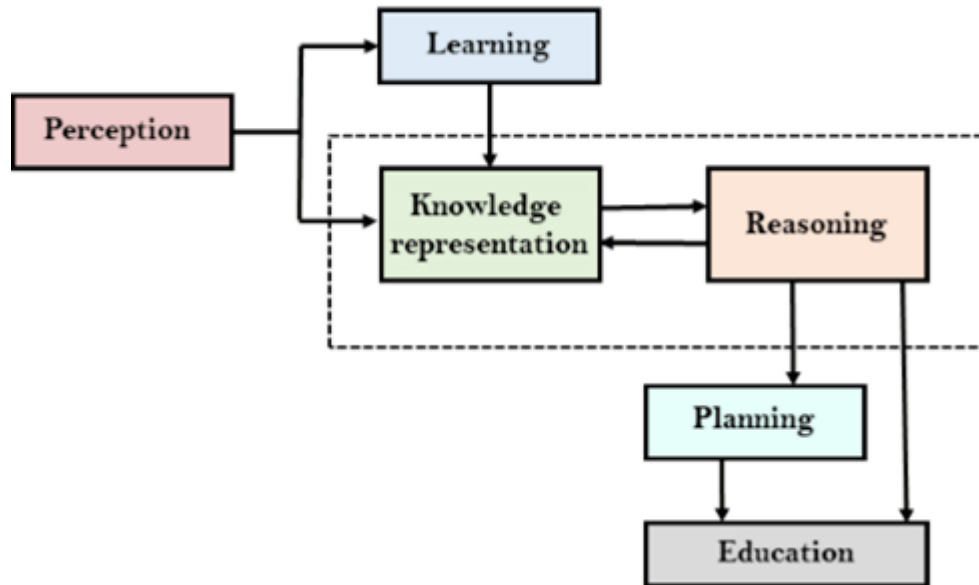


### **AI knowledge cycle:**

An Artificial intelligence system has the following components for displaying intelligent behavior:

- Perception
- Learning
- Knowledge Representation and Reasoning
- Planning
- Execution





The above diagram is showing how an AI system can interact with the real world and what components help it to show intelligence. AI system has Perception component by which it retrieves information from its environment. It can be visual, audio or another form of sensory input. The learning component is responsible for learning from data captured by Perception component. In the complete cycle, the main components are knowledge representation and Reasoning. These two components are involved in showing the intelligence in machine-like humans. These two components are independent with each other but also coupled together. The planning and execution depend on analysis of Knowledge representation and reasoning.

### **Approaches to knowledge representation:**

There are mainly four approaches to knowledge representation, which are given below:

#### **1. Simple relational knowledge:**

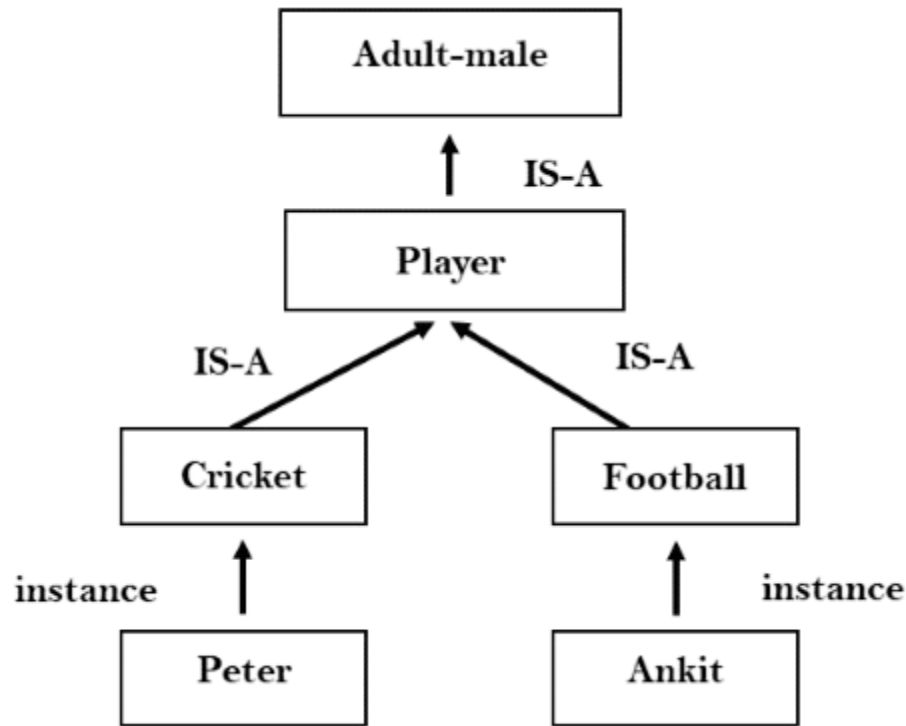
- It is the simplest way of storing facts which uses the relational method, and each fact about a set of the object is set out systematically in columns.
- This approach of knowledge representation is famous in database systems where the relationship between different entities is represented.
- This approach has little opportunity for inference.

**Example: The following is the simple relational knowledge representation.**

Player	Weight	Age
Player1	65	23
Player2	58	18
Player3	75	24

## 2. Inheritable knowledge:

- In the inheritable knowledge approach, all data must be stored into a hierarchy of classes.
- All classes should be arranged in a generalized form or a hierarchal manner.
- In this approach, we apply inheritance property.
- Elements inherit values from other members of a class.
- This approach contains inheritable knowledge which shows a relation between instance and class, and it is called instance relation.
- Every individual frame can represent the collection of attributes and its value.
- In this approach, objects and values are represented in Boxed nodes.
- We use Arrows which point from objects to their values.
- **Example:**



### 3. Inferential knowledge:

- Inferential knowledge approach represents knowledge in the form of formal logics.
- This approach can be used to derive more facts.
- It guaranteed correctness.
- **Example:** Let's suppose there are two statements:

a. Marcus is a man

b. All men are mortal  
Then it can represent as;

**man(Marcus)**

$\forall x = \text{man}(x) \text{ -----} \rightarrow \text{mortal}(x)$

### 4. Procedural knowledge:

- Procedural knowledge approach uses small programs and codes which describes how to do specific things, and how to proceed.
- In this approach, one important rule is used which is **If-Then rule**.

- In this knowledge, we can use various coding languages such as **LISP language** and **Prolog language**.
- We can easily represent heuristic or domain-specific knowledge using this approach.
- But it is not necessary that we can represent all cases in this approach.

### Requirements for knowledge Representation system:

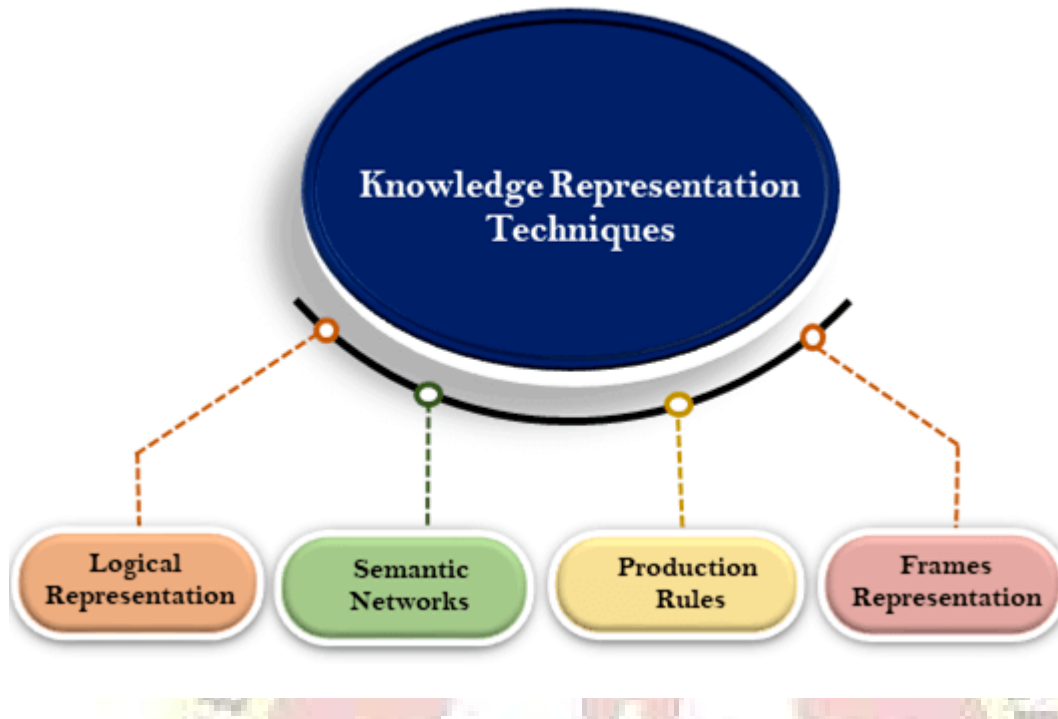
A good knowledge representation system must possess the following properties.

1. **1. Representational Accuracy:**  
KR system should have the ability to represent all kind of required knowledge.
2. **2. Inferential Adequacy:**  
KR system should have ability to manipulate the representational structures to produce new knowledge corresponding to existing structure.
3. **3. Inferential Efficiency:**  
The ability to direct the inferential knowledge mechanism into the most productive directions by storing appropriate guides.
4. **4. Acquisitional efficiency-** The ability to acquire the new knowledge easily using automatic methods.

### Techniques of knowledge representation

There are mainly four ways of knowledge representation which are given as follows:

1. Logical Representation
2. Semantic Network Representation
3. Frame Representation
4. Production Rules



## 1. Logical Representation

Logical representation is a language with some concrete rules which deals with propositions and has no ambiguity in representation. Logical representation means drawing a conclusion based on various conditions. This representation lays down some important communication rules. It consists of precisely defined syntax and semantics which supports the sound inference. Each sentence can be translated into logics using syntax and semantics.

### Syntax:

- Syntaxes are the rules which decide how we can construct legal sentences in the logic.
- It determines which symbol we can use in knowledge representation.
- How to write those symbols.

### Semantics:

- Semantics are the rules by which we can interpret the sentence in the logic.
- Semantic also involves assigning a meaning to each sentence.

Logical representation can be categorised into mainly two logics:

- a. Propositional Logics
- b. Predicate logics

*Note: We will discuss Propositional Logics and Predicate logics in later chapters.*

**Advantages of logical representation:**

1. Logical representation enables us to do logical reasoning.
2. Logical representation is the basis for the programming languages.

**Disadvantages of logical Representation:**

1. Logical representations have some restrictions and are challenging to work with.
2. Logical representation technique may not be very natural, and inference may not be so efficient.

*Note: Do not be confused with logical representation and logical reasoning as logical representation is a representation language and reasoning is a process of thinking logically.*

**2. Semantic Network Representation**

Semantic networks are alternative of predicate logic for knowledge representation. In Semantic networks, we can represent our knowledge in the form of graphical networks. This network consists of nodes representing objects and arcs which describe the relationship between those objects. Semantic networks can categorize the object in different forms and can also link those objects. Semantic networks are easy to understand and can be easily extended.

This representation consist of mainly two types of relations:

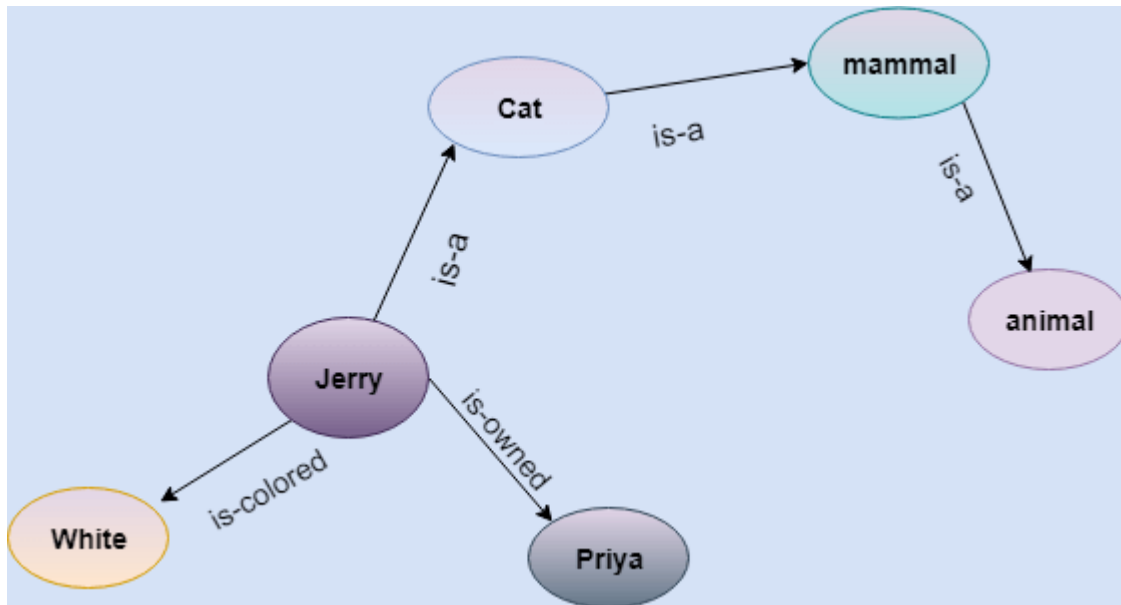
- a. IS-A relation (Inheritance)
- b. Kind-of-relation

**Example:** Following are some statements which we need to represent in the form of nodes and arcs.

**Statements:**

- a. Jerry is a cat.

- b. Jerry is a mammal
- c. Jerry is owned by Priya.
- d. Jerry is brown colored.
- e. All Mammals are animal.



In the above diagram, we have represented the different type of knowledge in the form of nodes and arcs. Each object is connected with another object by some relation.

#### Drawbacks in Semantic representation:

1. Semantic networks take more computational time at runtime as we need to traverse the complete network tree to answer some questions. It might be possible in the worst case scenario that after traversing the entire tree, we find that the solution does not exist in this network.
2. Semantic networks try to model human-like memory (Which has 10<sup>15</sup> neurons and links) to store the information, but in practice, it is not possible to build such a vast semantic network.
3. These types of representations are inadequate as they do not have any equivalent quantifier, e.g., for all, for some, none, etc.
4. Semantic networks do not have any standard definition for the link names.

5. These networks are not intelligent and depend on the creator of the system.

### Advantages of Semantic network:

1. Semantic networks are a natural representation of knowledge.
2. Semantic networks convey meaning in a transparent manner.
3. These networks are simple and easily understandable.

### 3. Frame Representation

A frame is a record like structure which consists of a collection of attributes and its values to describe an entity in the world. Frames are the AI data structure which divides knowledge into substructures by representing stereotypes situations. It consists of a collection of slots and slot values. These slots may be of any type and sizes. Slots have names and values which are called facets.

**Facets:** The various aspects of a slot is known as **Facets**. Facets are features of frames which enable us to put constraints on the frames. Example: IF-NEEDED facts are called when data of any particular slot is needed. A frame may consist of any number of slots, and a slot may include any number of facets and facets may have any number of values. A frame is also known as **slot-filter knowledge representation** in artificial intelligence.

Frames are derived from semantic networks and later evolved into our modern-day classes and objects. A single frame is not much useful. Frames system consist of a collection of frames which are connected. In the frame, knowledge about an object or event can be stored together in the knowledge base. The frame is a type of technology which is widely used in various applications including Natural language processing and machine visions.

#### Example: 1

Let's take an example of a frame for a book

**Slots**

**Filters**



<b>Title</b>	Artificial Intelligence
<b>Genre</b>	Computer Science
<b>Author</b>	Peter Norvig
<b>Edition</b>	Third Edition
<b>Year</b>	1996
<b>Page</b>	1152

### Example 2:

Let's suppose we are taking an entity, Peter. Peter is an engineer as a profession, and his age is 25, he lives in city London, and the country is England. So following is the frame representation for this:

Slots	Filter
<b>Name</b>	Peter
<b>Profession</b>	Doctor
<b>Age</b>	25
<b>Marital status</b>	Single
<b>Weight</b>	78

### Advantages of frame representation:

1. The frame knowledge representation makes the programming easier by grouping the related data.
2. The frame representation is comparably flexible and used by many applications in AI.
3. It is very easy to add slots for new attribute and relations.
4. It is easy to include default data and to search for missing values.
5. Frame representation is easy to understand and visualize.

### Disadvantages of frame representation:

1. In frame system inference mechanism is not be easily processed.
2. Inference mechanism cannot be smoothly proceeded by frame representation.
3. Frame representation has a much generalized approach.

## 4. Production Rules

Production rules system consist of (**condition, action**) pairs which mean, "If condition then action". It has mainly three parts:

- The set of production rules
- Working Memory
- The recognize-act-cycle

In production rules agent checks for the condition and if the condition exists then production rule fires and corresponding action is carried out. The condition part of the rule determines which rule may be applied to a problem. And the action part carries out the associated problem-solving steps. This complete process is called a recognize-act cycle.

The working memory contains the description of the current state of problems-solving and rule can write knowledge to the working memory. This knowledge match and may fire other rules.

If there is a new situation (state) generated, then multiple production rules will be fired together, this is called conflict set. In this situation, the agent needs to select a rule from these sets, and it is called a conflict resolution.

**Example:**

- **IF (at bus stop AND bus arrives) THEN action (get into the bus)**
- **IF (on the bus AND paid AND empty seat) THEN action (sit down).**
- **IF (on bus AND unpaid) THEN action (pay charges).**
- **IF (bus arrives at destination) THEN action (get down from the bus).**

**Advantages of Production rule:**

1. The production rules are expressed in natural language.
2. The production rules are highly modular, so we can easily remove, add or modify an individual rule.

**Disadvantages of Production rule:**

1. Production rule system does not exhibit any learning capabilities, as it does not store the result of the problem for the future uses.
2. During the execution of the program, many rules may be active hence rule-based production systems are inefficient.

**Propositional logic in Artificial intelligence**

Propositional logic (PL) is the simplest form of logic where all the statements are made by propositions. A proposition is a declarative statement which is either true or false. It is a technique of knowledge representation in logical and mathematical form.

**Example:**

1. a) It is Sunday.
2. b) The Sun rises from West (False proposition)
3. c)  $3+3=7$ (False proposition)
4. d) 5 is a prime number.

### Following are some basic facts about propositional logic:

- Propositional logic is also called Boolean logic as it works on 0 and 1.
- In propositional logic, we use symbolic variables to represent the logic, and we can use any symbol for a representing a proposition, such A, B, C, P, Q, R, etc.
- Propositions can be either true or false, but it cannot be both.
- Propositional logic consists of an object, relations or function, and **logical connectives**.
- These connectives are also called logical operators.
- The propositions and connectives are the basic elements of the propositional logic.
- Connectives can be said as a logical operator which connects two sentences.
- A proposition formula which is always true is called **tautology**, and it is also called a valid sentence.
- A proposition formula which is always false is called **Contradiction**.
- A proposition formula which has both true and false values is called
- Statements which are questions, commands, or opinions are not propositions such as "Where is Rohini", "How are you", "What is your name", are not propositions.

### Syntax of propositional logic:

The syntax of propositional logic defines the allowable sentences for the knowledge representation. There are two types of Propositions:

#### a. Atomic Propositions

#### b. Compound propositions

- **Atomic Proposition:** Atomic propositions are the simple propositions. It consists of a single proposition symbol. These are the sentences which must be either true or false.

### Example:

1. a)  $2+2$  is  $4$ , it is an atomic proposition as it is a **true** fact.
2. b) "The Sun is cold" is also a proposition as it is a **false** fact.

- **Compound proposition:** Compound propositions are constructed by combining simpler or atomic propositions, using parenthesis and logical connectives.

**Example:**

- a) "It is raining today, and street is wet."
- b) "Ankit is a doctor, and his clinic is in Mumbai."

**Logical Connectives:**

Logical connectives are used to connect two simpler propositions or representing a sentence logically. We can create compound propositions with the help of logical connectives. There are mainly five connectives, which are given as follows:

1. **Negation:** A sentence such as  $\neg P$  is called negation of P. A literal can be either Positive literal or negative literal.
2. **Conjunction:** A sentence which has  $\wedge$  connective such as,  $P \wedge Q$  is called a conjunction.  
**Example:** Rohan is intelligent and hardworking. It can be written as,  
**P= Rohan is intelligent,**  
**Q= Rohan is hardworking.  $\rightarrow P \wedge Q$ .**
3. **Disjunction:** A sentence which has  $\vee$  connective, such as  $P \vee Q$ . is called disjunction, where P and Q are the propositions.  
**Example: "Ritika is a doctor or Engineer",**  
 Here P= Ritika is Doctor. Q= Ritika is Doctor, so we can write it as  $P \vee Q$ .
4. **Implication:** A sentence such as  $P \rightarrow Q$ , is called an implication. Implications are also known as if-then rules. It can be represented as  
**If it is raining, then the street is wet.**  
 Let P= It is raining, and Q= Street is wet, so it is represented as  $P \rightarrow Q$
5. **Biconditional:** A sentence such as  $P \Leftrightarrow Q$  is a **Biconditional sentence, example If I am breathing, then I am alive**  
 P= I am breathing, Q= I am alive, it can be represented as  $P \Leftrightarrow Q$ .

Following is the summarized table for Propositional Logic Connectives:

Connective symbols	Word	Technical term	Example
$\wedge$	AND	Conjunction	$A \wedge B$
$\vee$	OR	Disjunction	$A \vee B$
$\rightarrow$	Implies	Implication	$A \rightarrow B$
$\leftrightarrow$	If and only if	Biconditional	$A \leftrightarrow B$
$\neg$ or $\sim$	Not	Negation	$\neg A$ or $\sim B$

### Truth Table:

In propositional logic, we need to know the truth values of propositions in all possible scenarios. We can combine all the possible combination with logical connectives, and the representation of these combinations in a tabular format is called **Truth table**. Following are the truth table for all logical connectives:



**For Negation:**

P	$\neg P$
True	False
False	True

**For Conjunction:**

P	Q	$P \wedge Q$
True	True	True
True	False	False
False	True	False
False	False	False

**For disjunction:**

P	Q	$P \vee Q$
True	True	True
False	True	True
True	False	True
False	False	False

**For Implication:**

P	Q	$P \rightarrow Q$
True	True	True
True	False	False
False	True	True
False	False	True

**For Biconditional:**

P	Q	$P \leftrightarrow Q$
True	True	True
True	False	False
False	True	False
False	False	True

### Truth table with three propositions:

We can build a proposition composing three propositions P, Q, and R. This truth table is made-up of 8n Tuples as we have taken three proposition symbols.

P	Q	R	$\neg R$	$P \vee Q$	$P \vee Q \rightarrow \neg R$
True	True	True	False	True	False
True	True	False	True	True	True
True	False	True	False	True	False
True	False	False	True	True	True
False	True	True	False	True	False
False	True	False	True	True	True
False	False	True	False	False	True
False	False	False	True	False	True

### Precedence of connectives:

Just like arithmetic operators, there is a precedence order for propositional connectors or logical operators. This order should be followed while evaluating a propositional problem. Following is the list of the precedence order for operators:

Precedence	Operators
First Precedence	Parenthesis
Second Precedence	Negation
Third Precedence	Conjunction(AND)
Fourth Precedence	Disjunction(OR)
Fifth Precedence	Implication



Six Precedence

Biconditional

*Note: For better understanding use parenthesis to make sure of the correct interpretations. Such as  $\neg R \vee Q$ , It can be interpreted as  $(\neg R) \vee Q$ .*

### Logical equivalence:

Logical equivalence is one of the features of propositional logic. Two propositions are said to be logically equivalent if and only if the columns in the truth table are identical to each other.

Let's take two propositions A and B, so for logical equivalence, we can write it as  $A \Leftrightarrow B$ . In below truth table we can see that column for  $\neg A \vee B$  and  $A \rightarrow B$ , are identical hence A is Equivalent to B

A	B	$\neg A$	$\neg A \vee B$	$A \rightarrow B$
T	T	F	T	T
T	F	F	F	F
F	T	T	T	T
F	F	T	T	T

### Properties of Operators:

- **Commutativity:**
  - $P \wedge Q = Q \wedge P$ , or
  - $P \vee Q = Q \vee P$ .
- **Associativity:**
  - $(P \wedge Q) \wedge R = P \wedge (Q \wedge R)$ ,
  - $(P \vee Q) \vee R = P \vee (Q \vee R)$
- **Identity element:**
  - $P \wedge \text{True} = P$ ,
  - $P \vee \text{True} = \text{True}$ .
- **Distributive:**
  - $P \wedge (Q \vee R) = (P \wedge Q) \vee (P \wedge R)$ .

- $P \vee (Q \wedge R) = (P \vee Q) \wedge (P \vee R)$ .
- **DE Morgan's Law:**
  - $\neg (P \wedge Q) = (\neg P) \vee (\neg Q)$
  - $\neg (P \vee Q) = (\neg P) \wedge (\neg Q)$ .
- **Double-negation elimination:**
  - $\neg (\neg P) = P$ .

### Limitations of Propositional logic:

- We cannot represent relations like ALL, some, or none with propositional logic.

Example:

- a. **All the girls are intelligent.**
- b. **Some apples are sweet.**

Propositional logic has limited expressive power.

In propositional logic, we cannot describe statements in terms of their properties or logical relationships.

### Rules of Inference in Artificial intelligence

#### Inference:

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

#### Inference rules:

Inference rules are the templates for generating valid arguments. Inference rules are applied to derive proofs in artificial intelligence, and the proof is a sequence of the conclusion that leads to the desired goal.

In inference rules, the implication among all the connectives plays an important role. Following are some terminologies related to inference rules:

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

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

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

Hence from the above truth table, we can prove that  $P \rightarrow Q$  is equivalent to  $\neg Q \rightarrow \neg P$ , and  $Q \rightarrow P$  is equivalent to  $\neg P \rightarrow \neg Q$ .

### Types of Inference rules:

#### 1. Modus Ponens:

The Modus Ponens rule is one of the most important rules of inference, and it states that if P and  $P \rightarrow Q$  is true, then we can infer that Q will be true. It can be represented as:

$$\text{Notation for Modus ponens: } \frac{P \rightarrow Q, P}{\therefore Q}$$

**Example:**

Statement-1: "If I am sleepy then I go to bed"  $\implies P \rightarrow Q$

Statement-2: "I am sleepy"  $\implies P$

Conclusion: "I go to bed."  $\implies Q$ .

Hence, we can say that, if  $P \rightarrow Q$  is true and  $P$  is true then  $Q$  will be true.

### Proof by Truth table:

P	Q	$P \rightarrow Q$
0	0	0
0	1	1
1	0	0
1	1	1

### 2. Modus Tollens:

The Modus Tollens rule state that if  $P \rightarrow Q$  is true and  $\neg Q$  is true, then  $\neg P$  will also true. It can be represented as:

$$\text{Notation for Modus Tollens: } \frac{P \rightarrow Q, \sim Q}{\sim P}$$

Statement-1: "If I am sleepy then I go to bed"  $\implies P \rightarrow Q$

Statement-2: "I do not go to the bed."  $\implies \sim Q$

Statement-3: Which infers that "I am not sleepy"  $\implies \sim P$

### Proof by Truth table:

P	Q	$\sim P$	$\sim Q$	$P \rightarrow Q$
0	0	1	1	1
0	1	1	0	1
1	0	0	1	0
1	1	0	0	1

### 3. Hypothetical Syllogism:

The Hypothetical Syllogism rule state that if  $P \rightarrow R$  is true whenever  $P \rightarrow Q$  is true, and  $Q \rightarrow R$  is true. It can be represented as the following notation:

**Example:**

**Statement-1:** If you have my home key then you can unlock my home.  $P \rightarrow Q$

**Statement-2:** If you can unlock my home then you can take my money.  $Q \rightarrow R$

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

**Proof by truth table:**

P	Q	R	$P \rightarrow Q$	$Q \rightarrow R$	$P \rightarrow R$
0	0	0	1	1	1
0	0	1	1	1	1
0	1	0	1	0	1
0	1	1	1	1	1
1	0	0	0	1	1
1	0	1	0	1	1
1	1	0	1	0	0
1	1	1	1	1	1

### 4. Disjunctive Syllogism:

The Disjunctive syllogism rule state that if  $P \vee Q$  is true, and  $\neg P$  is true, then  $Q$  will be true. It can be represented as:

$$\text{Notation of Disjunctive syllogism: } \frac{P \vee Q, \neg P}{Q}$$

**Example:**

**Statement-1:** Today is Sunday or Monday.  $\implies P \vee Q$

**Statement-2:** Today is not Sunday.  $\implies \neg P$

**Conclusion:** Today is Monday.  $\implies Q$

**Proof by truth-table:**

P	Q	$\neg P$	$P \vee Q$
0	0	1	0
0	1	1	1
1	0	0	1
1	1	0	1

**5. Addition:**

The Addition rule is one the common inference rule, and it states that If P is true, then  $P \vee Q$  will be true.

Notation of Addition:  $\frac{P}{P \vee Q}$

**Example:**

**Statement:** I have a vanilla ice-cream.  $\implies$  P

**Statement-2:** I have Chocolate ice-cream.

**Conclusion:** I have vanilla or chocolate ice-cream.  $\implies$  ( $P \vee Q$ )

**Proof by Truth-Table:**

P	Q	$P \vee Q$
0	0	0
1	0	1
0	1	1
1	1	1

**6. Simplification:**

The simplification rule state that if  $P \wedge Q$  is true, then **Q or P** will also be true. It can be represented as:

Notation of Simplification rule:  $\frac{P \wedge Q}{Q}$  Or  $\frac{P \wedge Q}{P}$

### Proof by Truth-Table:

P	Q	$P \wedge Q$
0	0	0
1	0	0
0	1	0
1	1	1

### 7. Resolution:

The Resolution rule state that if  $P \vee Q$  and  $\neg P \wedge R$  is true, then  $Q \vee R$  will also be true. **It can be represented as**

Notation of Resolution  $\frac{P \vee Q, \neg P \wedge R}{Q \vee R}$

### Proof by Truth-Table:

P	$\neg P$	Q	R	$P \vee Q$	$\neg P \wedge R$	$Q \vee R$
0	1	0	0	0	0	0
0	1	0	1	0	0	1
0	1	1	0	1	1	1
0	1	1	1	1	1	1
1	0	0	0	1	0	0
1	0	0	1	1	0	1
1	0	1	0	1	0	1
1	0	1	1	1	0	1

## UNIT III

### Planning and Acting in the Real World

#### What is planning in AI?

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

#### *Blocks-World planning problem*

- The blocks-world problem is known as **Sussman Anomaly**.
- Noninterleaved planners of the early 1970s were unable to solve this problem, hence it is considered as anomalous.
- When two subgoals G1 and G2 are given, a noninterleaved planner produces either a plan for G1 concatenated with a plan for G2, or vice-versa.
- In blocks-world problem, three blocks labeled as 'A', 'B', 'C' are allowed to rest on the flat surface. The given condition is that only one block can be moved at a time to achieve the goal.
- The start state and goal state are shown in the following diagram.

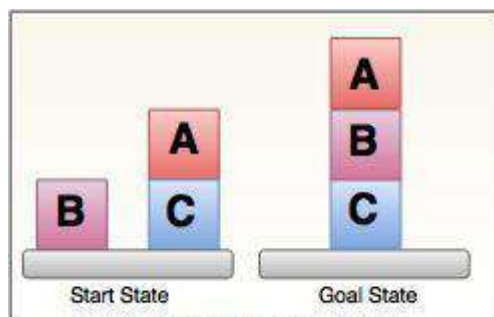


Fig: Blocks-World Planning Problem

#### *Components of Planning System*

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

- Choose the best rule for applying the next rule based on the best available heuristics.
- Apply the chosen rule for computing the new problem state.



- Detect when a solution has been found.
- Detect dead ends so that they can be abandoned and the system's effort is directed in more fruitful directions.
- Detect when an almost correct solution has been found.

### *Goal stack planning*

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

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

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

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

### Non-linear planning

This planning is used to set a goal stack and is included in the search space of all possible subgoal orderings. It handles the goal interactions by interleaving method.

#### **Advantage of non-Linear planning**

Non-linear planning may be an optimal solution with respect to plan length (depending on search strategy used).

#### **Disadvantages of Nonlinear planning**

- It takes larger search space, since all possible goal orderings are taken into consideration.

- Complex algorithm to understand.

### Algorithm

1. Choose a goal 'g' from the goalset
2. If 'g' does not match the state, then
  - Choose an operator 'o' whose add-list matches goal g
  - Push 'o' on the opstack
  - Add the preconditions of 'o' to the goalset
3. While all preconditions of operator on top of opstack are met in state
  - Pop operator o from top of opstack
  - state = apply(o, state)
  - plan = [plan; o]

## Probabilistic reasoning in Artificial intelligence

### Uncertainty:

Till now, we have learned knowledge representation using first-order logic and propositional logic with certainty, which means we were sure about the predicates. With this knowledge representation, we might write  $A \rightarrow B$ , which means if A is true then B is true, but consider a situation where we are not sure about whether A is true or not then we cannot express this statement, this situation is called uncertainty.

So to represent uncertain knowledge, where we are not sure about the predicates, we need uncertain reasoning or probabilistic reasoning.

### Causes of uncertainty:

Following are some leading causes of uncertainty to occur in the real world.

1. Information occurred from unreliable sources.
2. Experimental Errors
3. Equipment fault

4. Temperature variation
5. Climate change.

### **Probabilistic reasoning:**

Probabilistic reasoning is a way of knowledge representation where we apply the concept of probability to indicate the uncertainty in knowledge. In probabilistic reasoning, we combine probability theory with logic to handle the uncertainty.

We use probability in probabilistic reasoning because it provides a way to handle the uncertainty that is the result of someone's laziness and ignorance.

In the real world, there are lots of scenarios, where the certainty of something is not confirmed, such as "It will rain today," "behavior of someone for some situations," "A match between two teams or two players." These are probable sentences for which we can assume that it will happen but not sure about it, so here we use probabilistic reasoning.

### **Need of probabilistic reasoning in AI:**

- When there are unpredictable outcomes.
- When specifications or possibilities of predicates becomes too large to handle.
- When an unknown error occurs during an experiment.

In probabilistic reasoning, there are two ways to solve problems with uncertain knowledge:

- **Bayes' rule**
- **Bayesian Statistics**

*Note: We will learn the above two rules in later chapters.*

As probabilistic reasoning uses probability and related terms, so before understanding probabilistic reasoning, let's understand some common terms:

**Probability:** Probability can be defined as a chance that an uncertain event will occur. It is the numerical measure of the likelihood that an event will occur. The value of probability always remains between 0 and 1 that represent ideal uncertainties.

1.  $0 \leq P(A) \leq 1$ , where  $P(A)$  is the probability of an event A.

1.  $P(A) = 0$ , indicates total uncertainty in an event A.

1.  $P(A) = 1$ , indicates total certainty in an event A.

We can find the probability of an uncertain event by using the below formula.

$$\text{Probability of occurrence} = \frac{\text{Number of desired outcomes}}{\text{Total number of outcomes}}$$

- $P(\neg A)$  = probability of a not happening event.
- $P(\neg A) + P(A) = 1$ .

**Event:** Each possible outcome of a variable is called an event.

**Sample space:** The collection of all possible events is called sample space.

**Random variables:** Random variables are used to represent the events and objects in the real world.

**Prior probability:** The prior probability of an event is probability computed before observing new information.

**Posterior Probability:** The probability that is calculated after all evidence or information has taken into account. It is a combination of prior probability and new information.

**Conditional probability:**

Conditional probability is a probability of occurring an event when another event has already happened.

Let's suppose, we want to calculate the event A when event B has already occurred, "the probability of A under the conditions of B", it can be written as:

$$P(A|B) = \frac{P(A \cap B)}{P(B)}$$

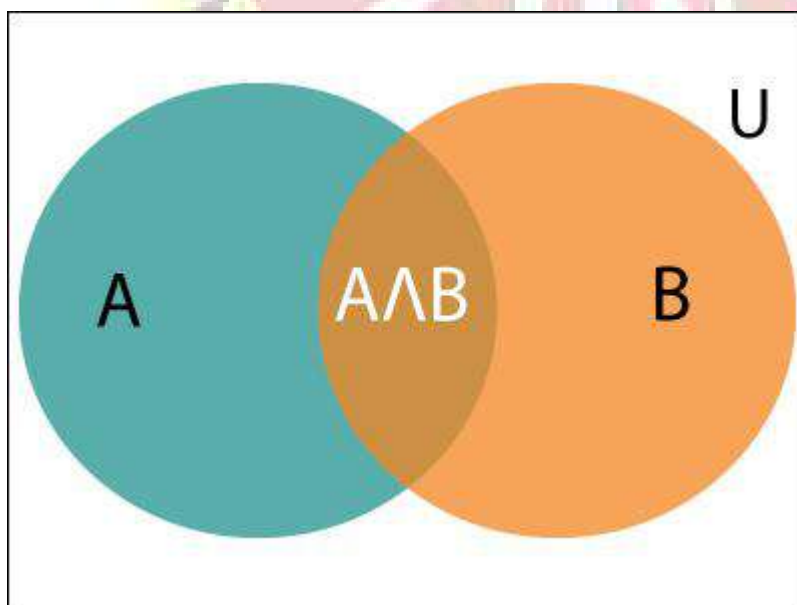
**Where  $P(A \cap B)$  = Joint probability of a and B**

**$P(B)$  = Marginal probability of B.**

If the probability of A is given and we need to find the probability of B, then it will be given as:

$$P(B|A) = \frac{P(A \cap B)}{P(A)}$$

It can be explained by using the below Venn diagram, where B is occurred event, so sample space will be reduced to set B, and now we can only calculate event A when event B is already occurred by dividing the probability of  $P(A \cap B)$  by  $P(B)$ .



**Example:**

In a class, there are 70% of the students who like English and 40% of the students who likes English and mathematics, and then what is the percent of students those who like English also like mathematics?

**Solution:**

Let, A is an event that a student likes Mathematics

B is an event that a student likes English.

$$P(A|B) = \frac{P(A \cap B)}{P(B)} = \frac{0.4}{0.7} = 57\%$$

**Hence, 57% are the students who like English also like Mathematics.**

**Bayes' theorem in Artificial intelligence**

**Bayes' theorem:**

Bayes' theorem is also known as **Bayes' rule**, **Bayes' law**, or **Bayesian reasoning**, which determines the probability of an event with uncertain knowledge.

In probability theory, it relates the conditional probability and marginal probabilities of two random events.

Bayes' theorem was named after the British mathematician **Thomas Bayes**. The **Bayesian inference** is an application of Bayes' theorem, which is fundamental to Bayesian statistics.

It is a way to calculate the value of  $P(B|A)$  with the knowledge of  $P(A|B)$ .

Bayes' theorem allows updating the probability prediction of an event by observing new information of the real world.

**Example:** If cancer corresponds to one's age then by using Bayes' theorem, we can determine the probability of cancer more accurately with the help of age.

Bayes' theorem can be derived using product rule and conditional probability of event A with known event B:

As from product rule we can write:

$$1. P(A \wedge B) = P(A|B) P(B) \text{ or}$$

Similarly, the probability of event B with known event A:

$$1. P(A \wedge B) = P(B|A) P(A)$$

Equating right hand side of both the equations, we will get:

$$P(A|B) = \frac{P(B|A) P(A)}{P(B)} \quad \dots(a)$$

The above equation (a) is called as **Bayes' rule** or **Bayes' theorem**. This equation is basic of most modern AI systems for **probabilistic inference**.

It shows the simple relationship between joint and conditional probabilities. Here,

$P(A|B)$  is known as **posterior**, which we need to calculate, and it will be read as Probability of hypothesis A when we have occurred an evidence B.

$P(B|A)$  is called the **likelihood**, in which we consider that hypothesis is true, then we calculate the probability of evidence.

$P(A)$  is called the **prior probability**, probability of hypothesis before considering the evidence

$P(B)$  is called **marginal probability**, pure probability of an evidence.

In the equation (a), in general, we can write  $P(B) = P(A) * P(B|A_i)$ , hence the Bayes' rule can be written as:

$$P(A_i|B) = \frac{P(A_i) \cdot P(B|A_i)}{\sum_{i=1}^k P(A_i) \cdot P(B|A_i)}$$

Where  $A_1, A_2, A_3, \dots, A_n$  is a set of mutually exclusive and exhaustive events.

### Applying Bayes' rule:

Bayes' rule allows us to compute the single term  $P(B|A)$  in terms of  $P(A|B)$ ,  $P(B)$ , and  $P(A)$ . This is very useful in cases where we have a good probability of these three terms and want to determine the fourth one. Suppose we want to perceive the effect of some unknown cause, and want to compute that cause, then the Bayes' rule becomes:

$$P(\text{cause}|\text{effect}) = \frac{P(\text{effect}|\text{cause}) P(\text{cause})}{P(\text{effect})}$$

### Example-1:

**Question:** what is the probability that a patient has diseases meningitis with a stiff neck?

### Given Data:

A doctor is aware that disease meningitis causes a patient to have a stiff neck, and it occurs 80% of the time. He is also aware of some more facts, which are given as follows:

- The Known probability that a patient has meningitis disease is 1/30,000.
- The Known probability that a patient has a stiff neck is 2%.

Let  $a$  be the proposition that patient has stiff neck and  $b$  be the proposition that patient has meningitis. , so we can calculate the following as:

$$P(a|b) = 0.8$$

$$P(b) = 1/30000$$



$$P(a) = .02$$

$$P(b|a) = \frac{P(a|b)P(b)}{P(a)} = \frac{0.8 * (\frac{1}{30000})}{0.02} = 0.0013333333.$$

Hence, we can assume that 1 patient out of 750 patients has meningitis disease with a stiff neck.

### Example-2:

**Question:** From a standard deck of playing cards, a single card is drawn. The probability that the card is king is  $4/52$ , then calculate posterior probability  $P(\text{King}|\text{Face})$ , which means the drawn face card is a king card.

**Solution:**

$$P(\text{king}|\text{face}) = \frac{P(\text{Face}|\text{king}) * P(\text{King})}{P(\text{Face})} \dots\dots(i)$$

$P(\text{king})$ : probability that the card is King =  $4/52 = 1/13$

$P(\text{face})$ : probability that a card is a face card =  $3/13$

$P(\text{Face}|\text{King})$ : probability of face card when we assume it is a king = 1

Putting all values in equation (i) we will get:

$$P(\text{king}|\text{face}) = \frac{1 * (\frac{1}{13})}{(\frac{3}{13})} = 1/3, \text{ it is a probability that a face card is a king card.}$$

### Application of Bayes' theorem in Artificial intelligence:

**Following are some applications of Bayes' theorem:**

- It is used to calculate the next step of the robot when the already executed step is given.

- Bayes' theorem is helpful in weather forecasting.
- It can solve the Monty Hall problem.

### **Bayesian Belief Network in artificial intelligence**

Bayesian belief network is key computer technology for dealing with probabilistic events and to solve a problem which has uncertainty. We can define a Bayesian network as:

"A Bayesian network is a probabilistic graphical model which represents a set of variables and their conditional dependencies using a directed acyclic graph."

It is also called a **Bayes network, belief network, decision network, or Bayesian model.**

Bayesian networks are probabilistic, because these networks are built from a **probability distribution**, and also use probability theory for prediction and anomaly detection.

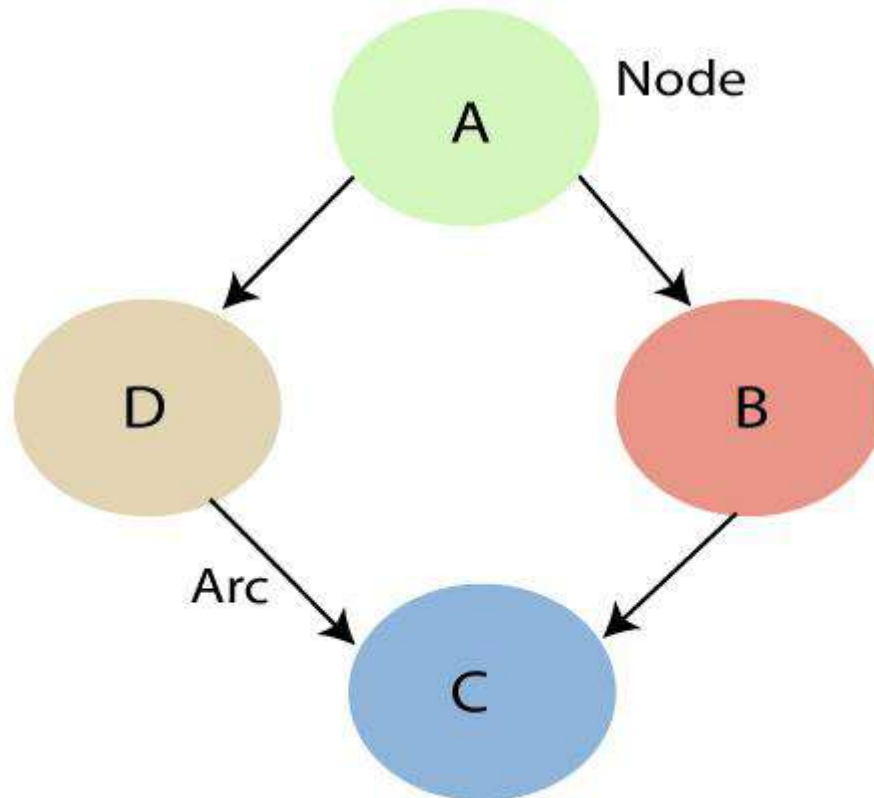
Real world applications are probabilistic in nature, and to represent the relationship between multiple events, we need a Bayesian network. It can also be used in various tasks including **prediction, anomaly detection, diagnostics, automated insight, reasoning, time series prediction, and decision making under uncertainty.**

Bayesian Network can be used for building models from data and experts opinions, and it consists of two parts:

- **Directed Acyclic Graph**
- **Table of conditional probabilities.**

The generalized form of Bayesian network that represents and solve decision problems under uncertain knowledge is known as an **Influence diagram.**

**A Bayesian network graph is made up of nodes and Arcs (directed links), where:**



- Each **node** corresponds to the random variables, and a variable can be **continuous** or **discrete**.
- **Arc or directed arrows** represent the causal relationship or conditional probabilities between random variables. These directed links or arrows connect the pair of nodes in the graph.

These links represent that one node directly influence the other node, and if there is no directed link that means that nodes are independent with each other

- **In the above diagram, A, B, C, and D are random variables represented by the nodes of the network graph.**
- **If we are considering node B, which is connected with node A by a directed arrow, then node A is called the parent of Node B.**
- **Node C is independent of node A.**

**Note: The Bayesian network graph does not contain any cyclic graph. Hence, it is known as a directed acyclic graph or DAG.**

The Bayesian network has mainly two components:

- **Causal Component**
- **Actual numbers**

Each node in the Bayesian network has condition probability distribution  $P(X_i | \text{Parent}(X_i))$ , which determines the effect of the parent on that node.

Bayesian network is based on Joint probability distribution and conditional probability. So let's first understand the joint probability distribution:

### **Joint probability distribution:**

If we have variables  $x_1, x_2, x_3, \dots, x_n$ , then the probabilities of a different combination of  $x_1, x_2, x_3 \dots x_n$ , are known as Joint probability distribution.

$P[x_1, x_2, x_3, \dots, x_n]$ , it can be written as the following way in terms of the joint probability distribution.

$$= P[x_1 | x_2, x_3, \dots, x_n] P[x_2, x_3, \dots, x_n]$$

$$= P[x_1 | x_2, x_3, \dots, x_n] P[x_2 | x_3, \dots, x_n] \dots P[x_{n-1} | x_n] P[x_n].$$

In general for each variable  $X_i$ , we can write the equation as:

$$P(X_i | X_{i-1}, \dots, X_1) = P(X_i | \text{Parents}(X_i))$$

### **Explanation of Bayesian network:**

Let's understand the Bayesian network through an example by creating a directed acyclic graph:

**Example:** Harry installed a new burglar alarm at his home to detect burglary. The alarm reliably responds at detecting a burglary but also responds for minor earthquakes. Harry has two

neighbors David and Sophia, who have taken a responsibility to inform Harry at work when they hear the alarm. David always calls Harry when he hears the alarm, but sometimes he got confused with the phone ringing and calls at that time too. On the other hand, Sophia likes to listen to high music, so sometimes she misses to hear the alarm. Here we would like to compute the probability of Burglary Alarm.

**Problem:**

**Calculate the probability that alarm has sounded, but there is neither a burglary, nor an earthquake occurred, and David and Sophia both called the Harry.**

**Solution:**

- The Bayesian network for the above problem is given below. The network structure is showing that burglary and earthquake is the parent node of the alarm and directly affecting the probability of alarm's going off, but David and Sophia's calls depend on alarm probability.
- The network is representing that our assumptions do not directly perceive the burglary and also do not notice the minor earthquake, and they also not confer before calling.
- The conditional distributions for each node are given as conditional probabilities table or CPT.
- Each row in the CPT must be sum to 1 because all the entries in the table represent an exhaustive set of cases for the variable.
- In CPT, a boolean variable with k boolean parents contains  $2^k$  probabilities. Hence, if there are two parents, then CPT will contain 4 probability values

**List of all events occurring in this network:**

- **Burglary (B)**
- **Earthquake(E)**
- **Alarm(A)**
- **David Calls(D)**

- Sophia calls(S)

We can write the events of problem statement in the form of probability:  $P[D, S, A, B, E]$ , can rewrite the above probability statement using joint probability distribution:

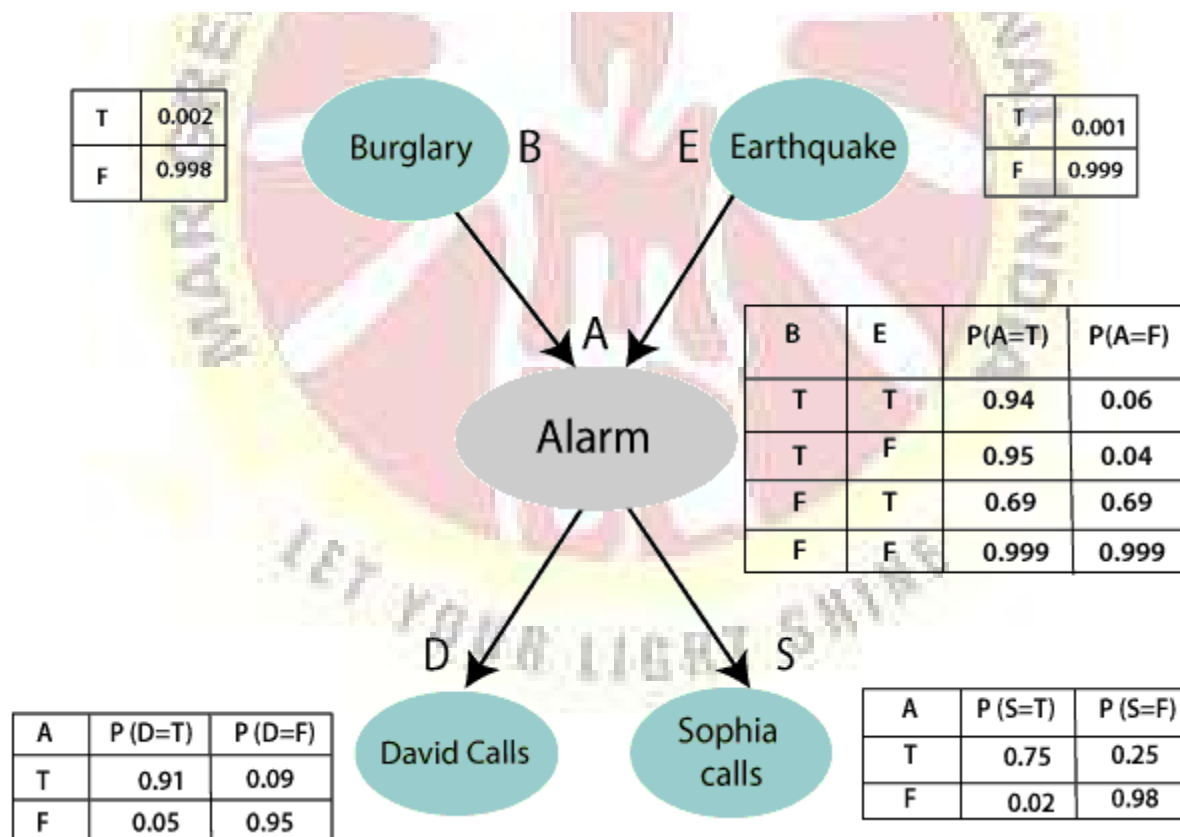
$$P[D, S, A, B, E] = P[D | S, A, B, E] \cdot P[S, A, B, E]$$

$$= P[D | S, A, B, E] \cdot P[S | A, B, E] \cdot P[A, B, E]$$

$$= P[D | A] \cdot P[S | A, B, E] \cdot P[A, B, E]$$

$$= P[D | A] \cdot P[S | A] \cdot P[A | B, E] \cdot P[B, E]$$

$$= P[D | A] \cdot P[S | A] \cdot P[A | B, E] \cdot P[B | E] \cdot P[E]$$



Let's take the observed probability for the Burglary and earthquake component:

$P(B = \text{True}) = 0.002$ , which is the probability of burglary.

$P(B = \text{False}) = 0.998$ , which is the probability of no burglary.

$P(E = \text{True}) = 0.001$ , which is the probability of a minor earthquake

$P(E = \text{False}) = 0.999$ , Which is the probability that an earthquake not occurred.

We can provide the conditional probabilities as per the below tables:

#### **Conditional probability table for Alarm A:**

The Conditional probability of Alarm A depends on Burglar and earthquake:

B	E	P(A= True)	P(A= False)
True	True	0.94	0.06
True	False	0.95	0.04
False	True	0.31	0.69
False	False	0.001	0.999

#### **Conditional probability table for David Calls:**

The Conditional probability of David that he will call depends on the probability of Alarm.

A	P(D= True)	P(D= False)
True	0.91	0.09
False	0.05	0.95

### Conditional probability table for Sophia Calls:

The Conditional probability of Sophia that she calls is depending on its Parent Node "Alarm."

A	P(S= True)	P(S= False)
True	0.75	0.25
False	0.02	0.98

From the formula of joint distribution, we can write the problem statement in the form of probability distribution:

$$P(S, D, A, \neg B, \neg E) = P(S|A) * P(D|A) * P(A|\neg B \wedge \neg E) * P(\neg B) * P(\neg E).$$

$$= 0.75 * 0.91 * 0.001 * 0.998 * 0.999$$

$$= \mathbf{0.00068045}.$$

**Hence, a Bayesian network can answer any query about the domain by using Joint distribution.**

### The semantics of Bayesian Network:

There are two ways to understand the semantics of the Bayesian network, which is given below:

#### 1. To understand the network as the representation of the Joint probability distribution.

It is helpful to understand how to construct the network.

#### 2. To understand the network as an encoding of a collection of conditional independence statements.

It is helpful in designing inference procedure.

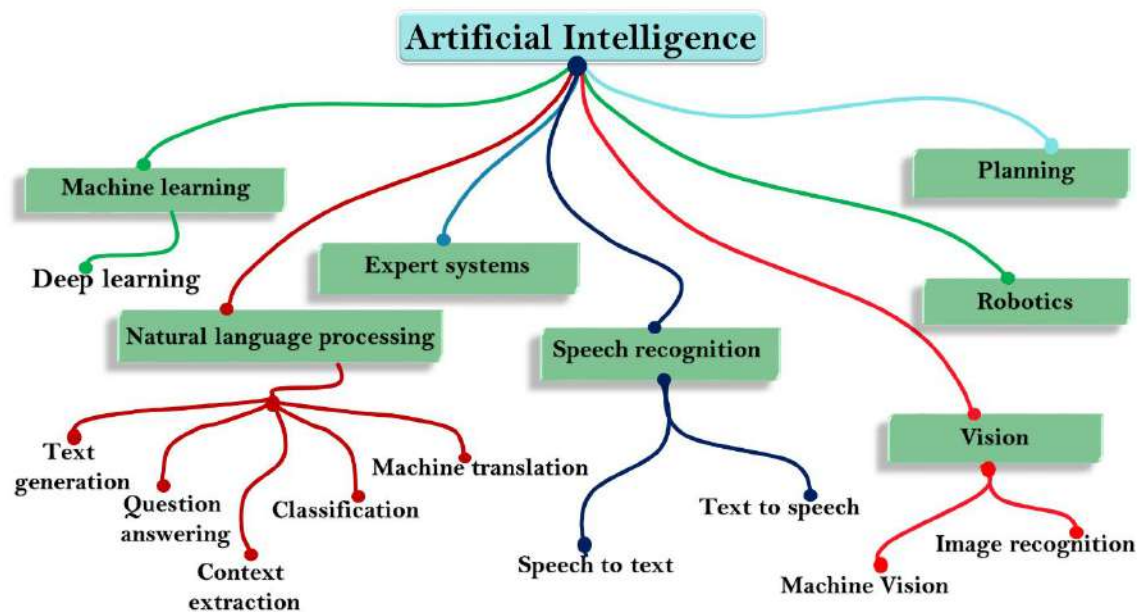


## Subsets of Artificial Intelligence

Till now, we have learned about what is AI, and now we will learn in this topic about various subsets of AI. Following are the most common subsets of AI:

- Machine Learning
- Deep Learning
- Natural Language processing
- Expert System
- Robotics
- Machine Vision
- Speech Recognition

*Note: Among all of the above, Machine learning plays a crucial role in AI. Machine learning and deep learning are the ways of achieving AI in real life.*

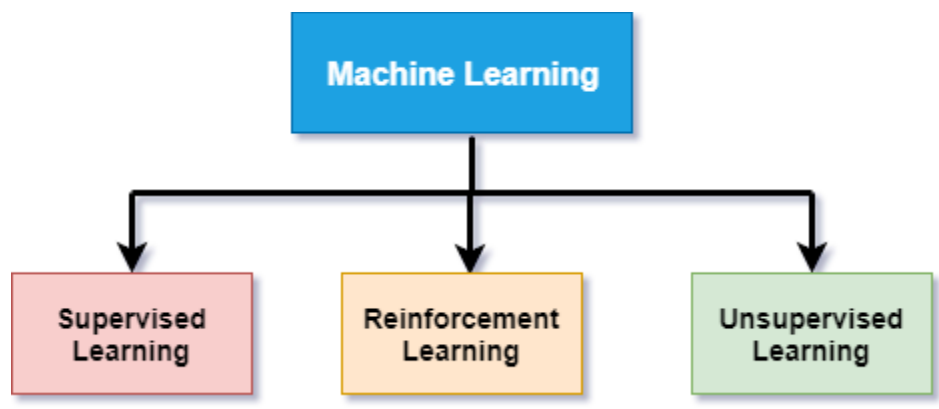


## Machine Learning

Machine learning is a part of AI which provides intelligence to machines with the ability to automatically learn with experiences without being explicitly programmed.

- It is primarily concerned with the design and development of algorithms that allow the system to learn from historical data.
- Machine Learning is based on the idea that machines can learn from past data, identify patterns, and make decisions using algorithms.
- Machine learning algorithms are designed in such a way that they can learn and improve their performance automatically.
- Machine learning helps in discovering patterns in data.

### Types of Machine Learning



Machine learning can be subdivided into the main three types:

- **Supervised learning:**  
Supervised learning is a type of machine learning in which machine learn from known datasets (set of training examples), and then predict the output. A supervised learning agent needs to find out the function that matches a given sample set. Supervised learning further can be classified into two categories of algorithms:

- Classifications**
- Regression**

#### **Reinforcement learning:**

Reinforcement learning is a type of learning in which an AI agent is trained by giving some commands, and on each action, an agent gets a reward as a feedback. Using these feedbacks, agent improves its performance. Reward feedback can be positive or negative which means on each good action, agent receives a

positive reward while for wrong action, it gets a negative reward. Reinforcement learning is of two types:

- . **Positive Reinforcement learning**
- a. **Negative Reinforcement learning**

### **Unsupervised learning:**

Unsupervised learning is associated with learning without supervision or training. In unsupervised learning, the algorithms are trained with data which is neither labeled nor classified. In unsupervised learning, the agent needs to learn from patterns without corresponding output values.

Unsupervised learning can be classified into two categories of algorithms:

- . **Clustering**
- a. **Association**

### **Natural Language processing**

Natural language processing is a subfield of computer science and artificial intelligence. NLP enables a computer system to understand and process human language such as English.

NLP plays an important role in AI as without NLP, AI agent cannot work on human instructions, but with the help of NLP, we can instruct an AI system on our language. Today we are all around AI, and as well as NLP, we can easily ask Siri, Google or Cortana to help us in our language.

Natural language processing application enables a user to communicate with the system in their own words directly.

The Input and output of NLP applications can be in two forms:

- o **Speech**
- o **Text**

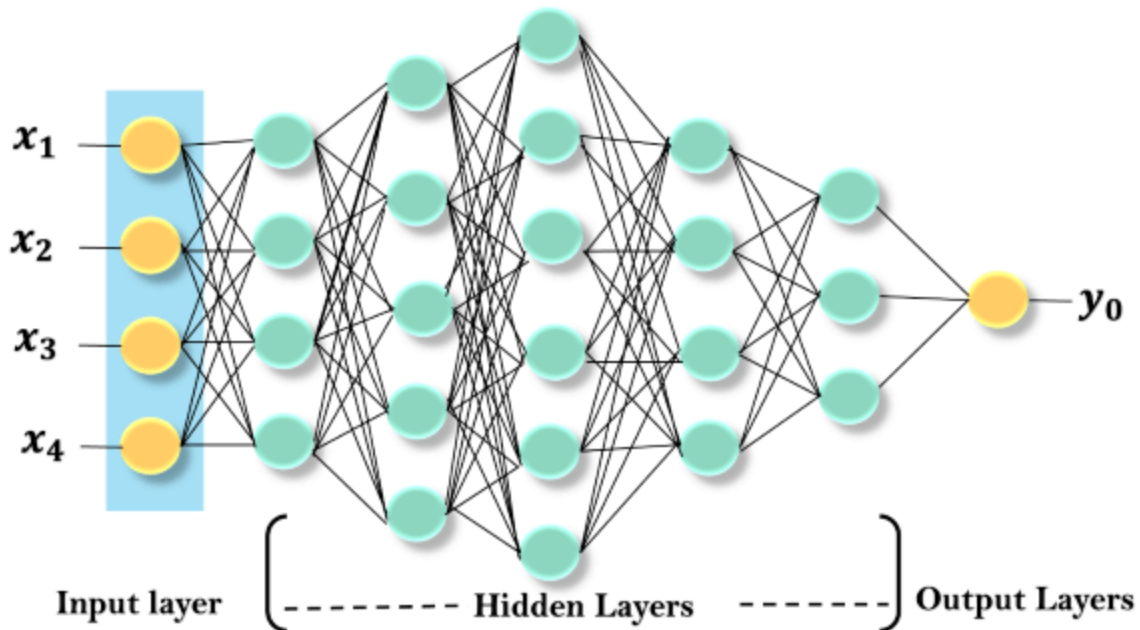
## Deep Learning

Deep learning is a subset of machine learning which provides the ability to machine to perform human-like tasks without human involvement. It provides the ability to an AI agent to mimic the human brain. Deep learning can use both supervised and unsupervised learning to train an AI agent.

- Deep learning is implemented through neural networks architecture hence also called a **deep neural network**.
- Deep learning is the primary technology behind self-driving cars, speech recognition, image recognition, automatic machine translation, etc.
- The main challenge for deep learning is that it requires lots of data with lots of computational power.

### How deep learning works:

- **Deep Learning Algorithms work on deep neural networks, so it is called deep learning. These deep neural networks are made of multiple layers.**
- **The first layer is called an Input layer, the last layer is called an output layer, and all layers between these two layers are called hidden layers.**
- **In the deep neural network, there are multiple hidden layers, and each layer is composed of neurons. These neurons are connected in each layer.**
- **The input layer receives input data, and the neurons propagate the input signal to its above layers.**
- **The hidden layers perform mathematical operations on inputs, and the performed data forwarded to the output layer.**
- **The output layer returns the output to the user.**



### Expert Systems

- An expert system is an application of artificial intelligence. In artificial intelligence, expert systems are the computer programs that rely on obtaining the knowledge of human experts and programming that knowledge into a system.
- Expert systems emulate the decision-making ability of human experts. These systems are designed to solve the complex problem through bodies of knowledge rather than conventional procedural code.
- One of the examples of an expert system is a Suggestion for the spelling error while typing in the Google search box.
- Following are some characteristics of expert systems:
  - High performance
  - Reliable
  - Highly responsive
  - Understandable

## Robotics

- Robotics is a branch of artificial intelligence and engineering which is used for designing and manufacturing of robots.
- Robots are the programmed machines which can perform a series of actions automatically or semi-automatically.
- AI can be applied to robots to make intelligent robots which can perform the task with their intelligence. AI algorithms are necessary to allow a robot to perform more complex tasks.
- Nowadays, AI and machine learning are being applied on robots to manufacture intelligent robots which can also interact socially like humans. One of the best examples of AI in robotics is Sophia robot.

## Machine Vision

- Machine vision is an application of computer vision which enables a machine to recognize the object.
- Machine vision captures and analyses visual information using one or more video cameras, analog-to-digital conversions, and digital signal processing.
- Machine vision systems are programmed to perform narrowly defined tasks such as counting objects, reading the serial number, etc.
- Computer systems do not see in the same way as human eyes can see, but it is also not bounded by human limitations such as to see through the wall.
- With the help of machine learning and machine vision, an AI agent can be able to see through walls.

## Speech Recognition:

Speech recognition is a technology which enables a machine to understand the spoken language and translate into a machine-readable format. It can also be said as automatic Speech recognition and computer speech recognition. **It is a way to talk with a computer, and on the basis of that command, a computer can perform a specific task.**

There is some speech recognition software which has a limited vocabulary of words and phrase. This software requires unambiguous spoken language to understand and perform specific task. Today's there are various software or devices which contains speech recognition technology such as Cortana, Google virtual assistant, Apple Siri, etc.

We need to train our speech recognition system to understand our language. In previous days, these systems were only designed to convert the speech to text, but now there are various devices which can directly convert speech into commands.

Speech recognition systems can be used in the following areas:

- **System control or navigation system**
- **Industrial application**
- **Voice dialing system**

There are two types of speech recognition

1. **Speaker Dependent**
2. **Speaker Independent**

### **Simple Decision Making**

A **decision network** (also called an **influence diagram**) is a graphical representation of a finite sequential decision problem. Decision networks extend belief networks to include decision variables and utility. A decision network extends the [single-stage decision network](#) to allow for sequential decisions.

In particular, a **decision network** is a directed acyclic graph (DAG) with chance nodes, decision nodes, and a utility node. This extends single-stage decision networks by allowing both chance nodes and decision nodes to be parents of decision nodes. Arcs coming into decision nodes represent the information that will be available when the decision is made. Arcs coming into chance nodes represents probabilistic dependence. Arcs coming into the utility node represent what the utility depends on.

A **no-forgetting agent** is an agent whose decisions are totally ordered, and the agent remembers its previous decisions and any information that was available to a previous decision. A **no-forgetting decision network** is a decision network in which the decision nodes are totally ordered and, if decision node  $D_i$  is before  $D_j$  in the total ordering, then  $D_i$  is a parent of  $D_j$ , and any parent of  $D_i$  is also a parent of  $D_j$ . Thus, any information available to  $D_i$  is available to  $D_j$ , and the action chosen for decision  $D_i$  is part of the information available at decision  $D_j$ . The no-forgetting condition is sufficient to make sure that the following definitions make sense and that the following algorithms work.

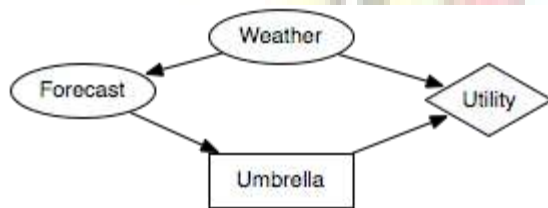


Figure 9.7: Decision network for decision of whether to take an umbrella

**Example 9.11:** Figure 9.7 shows a simple decision network for a decision of whether the agent should take an umbrella when it goes out. The agent's utility depends on the weather and whether it takes an umbrella. However, it does not get to observe the weather. It only gets to observe the forecast. The forecast probabilistically depends on the weather.

As part of this network, the designer must specify the domain for each random variable and the domain for each decision variable. Suppose the random variable *Weather* has domain  $\{norain, rain\}$ , the random variable *Forecast* has domain  $\{sunny, rainy, cloudy\}$ , and the decision variable *Umbrella* has domain  $\{takeIt, leaveIt\}$ . There is no domain associated with the utility node. The designer also must specify the probability of the random variables given their parents. Suppose  $P(Weather)$  is defined by

$$P(Weather=rain)=0.3.$$



$P(\text{Forecast}|\text{Weather})$  is given by

<i>Weather</i>	<i>Forecast</i>	Probability
<i>norain</i>	<i>sunny</i>	0.7
<i>norain</i>	<i>cloudy</i>	0.2
<i>norain</i>	<i>rainy</i>	0.1
<i>rain</i>	<i>sunny</i>	0.15
<i>rain</i>	<i>cloudy</i>	0.25
<i>rain</i>	<i>rainy</i>	0.6

Suppose the utility function,  $Utility(\text{Weather}, \text{Umbrella})$ , is

<i>Weather</i>	<i>Umbrella</i>	Utility
<i>norain</i>	<i>takeIt</i>	20
<i>norain</i>	<i>leaveIt</i>	100
<i>rain</i>	<i>takeIt</i>	70
<i>rain</i>	<i>leaveIt</i>	0

There is no table specified for the *Umbrella* decision variable. It is the task of the planner to determine which value of *Umbrella* to select, depending on the forecast.

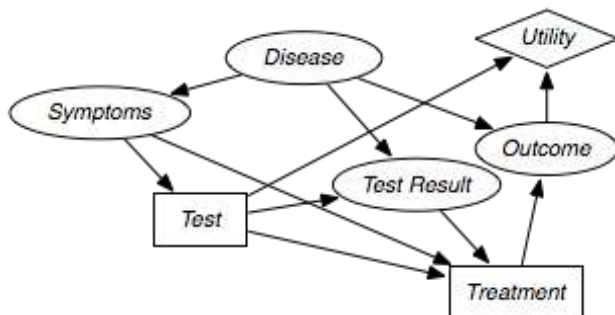


Figure 9.8: Decision network for diagnosis

**Example 9.12:** Figure 9.8 shows a decision network that represents the scenario of Example 9.10. The symptoms depend on the disease. What test to perform is decided based on the symptoms. The test result depends on the disease and the test performed. The treatment decision is based on the symptoms, the test performed, and the test result. The outcome depends on the disease and the treatment. The utility depends on the costs and the side effects of the test and on the outcome.

Note that the diagnostic assistant that is deciding on the tests and the treatments never actually finds out what disease the patient has, unless the test result is definitive, which it typically is not.

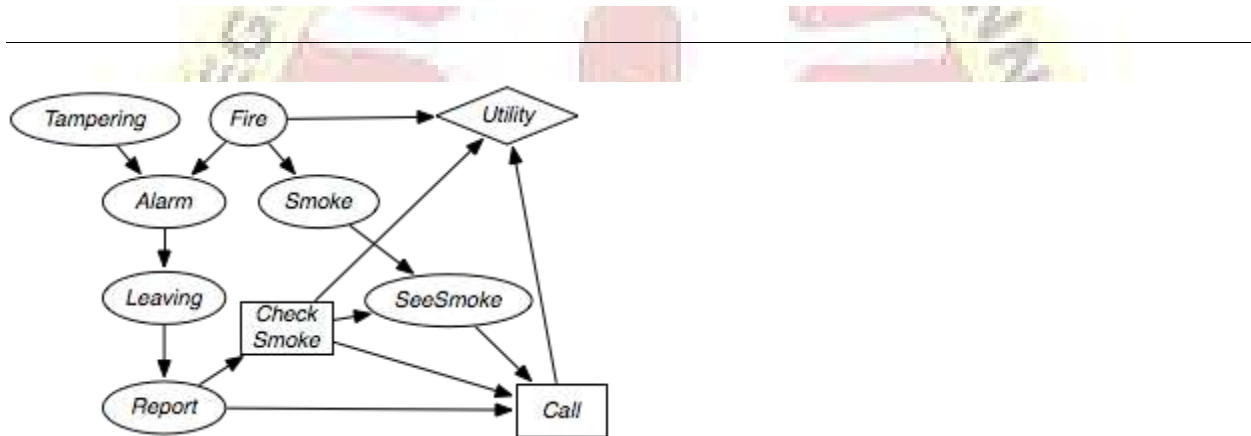


Figure 9.9: Decision network for the alarm problem

**Example 9.13:** Figure 9.9 gives a decision network that is an extension of the belief network of Figure 6.1. The agent can receive a report of people leaving a building and has to decide whether or not to call the fire department. Before calling, the agent can check for smoke, but this has some cost associated with it. The utility depends on whether it calls, whether there is a fire, and the cost associated with checking for smoke.

In this sequential decision problem, there are two decisions to be made. First, the agent must decide whether to check for smoke. The information that will be available when it makes this decision is whether there is a report of people leaving the building. Second, the agent must

decide whether or not to call the fire department. When making this decision, the agent will know whether there was a report, whether it checked for smoke, and whether it can see smoke. Assume that all of the variables are binary.

The information necessary for the decision network includes the conditional probabilities of the belief network and

- $P(\text{SeeSmoke}|\text{Smoke},\text{CheckSmoke})$ ; how seeing smoke depends on whether the agent looks for smoke and whether there is smoke. Assume that the agent has a perfect sensor for smoke. It will see smoke if and only if it looks for smoke and there is smoke. [See [Exercise 9.6](#).]
- $Utility(\text{CheckSmoke},\text{Fire},\text{Call})$ ; how the utility depends on whether the agent checks for smoke, whether there is a fire, and whether the fire department is called. [Figure 9.10](#) provides this utility information.

---

<i>CheckSmoke</i>	<i>Fire</i>	<i>Call</i>	<i>Utility</i>
yes	yes	call	-220
yes	yes	do not call	-5020
yes	no	call	-220
yes	no	do not call	-20
no	yes	call	-200
no	yes	do not call	-5000
no	no	call	-200
no	no	do not call	0

Figure 9.10: Utility for alarm decision network

---

This utility function expresses the cost structure that calling has a cost of 200, checking has a cost of 20, but not calling when there is a fire has a cost of 5,000. The utility is the negative of the cost.

The integration of artificial intelligence (AI) into acute care brings a new source of intellectual thought to the bedside. This offers great potential for synergy between AI systems and the human intellect already delivering care. This much needed help should be embraced, if proven effective. However, there is a risk that the present role of physicians and nurses as the primary arbiters of acute care in hospitals may be overtaken by computers. While many argue that this transition is inevitable, the process of developing a formal plan to prevent the need to pass control of patient care to computers should not be further delayed.

The first step in the interdiction process is to recognize; the limitations of existing hospital protocols, why we need AI in acute care, and finally how the focus of medical decision making will change with the integration of AI based analysis. The second step is to develop a strategy for changing the focus of medical education to empower physicians to maintain oversight of AI. Physicians, nurses, and experts in the field of safe hospital communication must control the transition to AI integrated care because there is significant risk during the transition period and much of this risk is subtle, unique to the hospital environment, and outside the expertise of AI designers.

AI is needed in acute care because AI detects complex relational time-series patterns within datasets and this level of analysis transcends conventional threshold based analysis applied in hospital protocols in use today. For this reason medical education will have to change to provide healthcare workers with the ability to understand and over-read relational time pattern centered communications from AI. Medical education will need to place less emphasis on threshold decision making and a greater focus on detection, analysis, and the pathophysiologic basis of relational time patterns. This should be an early part of a medical student's education because this is what their hospital companion (the AI) will be doing.

Effective communication between human and artificial intelligence requires a common pattern centered knowledge base. Experts in safety focused human to human communication in hospitals should lead during this transition process.

## Background

---

### **Facing the challenge from Silicon Valley**

Three thousand years ago the perceived power of the human brain to affect survival from illness was hyperbolized in Asclepius, the Greek god of medicine. Asclepius was credited with such a powerful intellect that he altered the ratio of living to dead.

While mortal physicians have never matched his success, for thousands of years patients have placed their confidence in the intellect of physicians for medical diagnosis and care. However, the present role of physicians and nurses as preeminent diagnosticians and providers of care may soon be overtaken by computers. While many argue that this transition is inevitable, physicians have not yet developed a formal plan to respond to the challenge from Silicon Valley. This is a momentous time and the process of developing a formal plan to prevent the need to pass the Caduceus should not be further delayed.

The first step in the prevention of loss of the position of physicians and nurses as preeminent overseers of hospital care is to understand the present limitations of medical diagnostics in the acute care environment and in medical education which have driven the need for AI integration.

### **The present state of acute care decision making**

A computer programmer examining the present threshold based hospital protocols with the intent to develop algorithms for managing care might quickly conclude that automation of acute care diagnostics and treatment would be easy to implement. The reason for this is that present hospital protocols are based on twentieth century threshold decision making [1] and are generally quite simple. In an example, it might appear to a computer programmer that all that is required to diagnose and treat sepsis is an indication that infection is suspected, a simple threshold breach detection algorithm [2] and a branching set of treatment rules.

However expert clinicians know that these simple protocols are not indicative of the true levels of acute care complexity [3, 4]. The randomized controlled trials (RCT) which use the threshold rules applied in hospital protocols as unified standards for an entire population are subject to marked heterogeneous treatment effects (HTE) [5]. Such trials provide evidence of the average treatment effect on the group under test as a whole but not whether the treatment used in the RCT will be beneficial or harmful to the instant patient under care. It therefore logically follows that no protocol, no matter how well supported by RCT, can be applied without expert oversight provided by either by a human or AI to protect patients from harm.

For this reason designers of automated systems must recognize that the true complexity of diagnosis and management of adverse conditions, such as sepsis, resides in the portion of diagnosis and care provided by nuanced expert oversight which is difficult to study and reproduce. If the patient is worsening despite adherence to the standard protocol, a physician with naïve trust in RCT may adhere to the guidelines thinking she must stay the course. In the alternative the expert physician knows to detect and track worsening, knows the potential for HTE, and modifies care off protocol if necessary. In addition to responding to worsening, expert clinical nuance is also applied to change diagnosis and/or alter the care in the presence of an unusual presentation, rare cases, and less common mixes of overlapping diseases.

Figure 1 illustrates the present state of hospital care based on uniform protocols with physicians and nurses operating as expert overseers, protecting patients from the hazards of oversimplified threshold based decisions. Physicians and nurses can provide oversight because the decision process of the protocols are transparent so, using their own intellect, they are able to modify the diagnosis and treatment as needed in real-time.

**Fig. 1**

The present state of nuanced threshold decision making. Threshold protocols are simple and transparent so nuanced care with complex decisions outside the protocols can be provided by the nurse/physician when needed

[Full size image](#)

The integration of artificial intelligence into acute care

---

### **AI and the acute care environment**

The future of medical AI extends far beyond the interpretation of medical images, pathology slides and radiographs. AI is being developed to detect critical, highly complex and time dependent conditions such as adverse drug reactions and sepsis [6, 7] in acute care environments where timely nuanced communication is pivotal. It is likely that within 5 years, AI based protocols will replace many of the threshold based protocols which are presently providing acute care diagnostic and treatment decisions.

One advantage of AI is that it can analyze more relationally complex portions of a patient's dataset. However, a major disadvantage is that the complex decision processing of the AI may be substantially opaque if not designed to provide transparency and nuanced communication. Effective communication from an AI must not be inferior to communication from a human.

However the need for detailed real-time communication is even more acute when the adverse clinical conditions under investigation and care are highly complex and rapidly progressing critical conditions. For this reason, the introduction of AI into the acute care environment should be preceded by detailed consideration of how AI, if not properly implemented, may cause harm by adversely affecting communication as well as the role of the physician and nurse as expert overseers of care.

### **The hazards of black-box artificial intelligence in acute care**

In the worst case, complex analytics may be provided by AI without disclosure of the data used to make the decision or the analytic processes applied. This is often called "black-box AI". This approach is reasonable in acute care only if expert human oversight will not be beneficial even if the patient is worsening. One example of inadequate black box AI would be the application of an AI based protocol which, after an opaque analysis of a complex data set, provides only an output stating "Sepsis detected--severity score 20". This limited output would constitute a

communication error by human standards because it does not communicate the relational time patterns (RTP) of laboratory tests and vitals detected, the other factors considered, how the patterns were combined, which component patterns are rapidly worsening, and how the overall state of the sepsis pattern has progressed over time for example in relation to treatment or to a potentially inciting event such as a surgery. This level of communication would be required if the detection was performed by a human expert so the AI must deliver equivalent or better communication.

While black box AI is clearly unacceptable in acute care, the standards for AI transparency and for timely communication by AI of the details of its decision process have not been determined. Yet it is clear that AI should meet the minimum standards of detail required for physician to physician handoff.

Figure 2 shows that the use of Black Box AI (which has not been programmed to communicate the nuanced details of the decision process) renders a state wherein the AI stands alone. Oversight by the expert is not facilitated because she cannot see the pattern combinations detected by the AI.

**Fig. 2**

The bedside state with “Black Box AI” based decision making. Complex decisions made by AI without transparency or detailed timely communication of the factors considered and how these factors were combined to render decision process. If the patient worsens under the care of the AI the handoff to the physician will be blind

[Full size image](#)



### **Medical diagnostic analysis is more complex than in many other environments**

One problem with AI use in acute care which may not be readily apparent to those with a limited understanding of the medical domain is the common delay between the decision and the result which occurs responsive to the decision. Often it is not clear that a medical decision was wrong for a given patient's condition until many days later when complications, recovery failure, or worsening occurs. Contrast this to AI based autonomous driving. Here, a human in the car can rapidly intervene because the correctness of the decisions made by the AI are generally immediately transparent to the overseeing driver. The immediacy of feedback simplifies the training processes of AI based autonomous driving and allows the use of black box AI because the decision process is largely irrelevant and can be opaque to the driver who is only concerned with the result which is promptly apparent. By comparison, in medical care, where the correctness of the decision is often not immediately obvious, the clinician responsible for the patient cannot simply wait with faith to see the outcome. Rather the clinician needs to be able to see the decision making process itself in fine relational detail and in real-time (before the outcome) to be sure it applies to the complexities and comorbidities of the instant patient under care. Here we can easily see the need for a new focus of medical education because bedside transparency of AI is not useful if the physician is not trained to be able to interpret the RTP detected by the AI.

### **The handoff from an artificial intelligence system to a physician**

The addition of AI brings new communication challenges to a complex hospital environment already associated with a high error rate. Communication errors, including those associated with patient handoffs between multiple humans, have been cited by the Joint Commission as the root cause of up to 60% of adverse events [8, 9]. Yet any action which moves care from one caregiver to another is a handoff. This is true even if a computer is the diagnostician and is now handing off the care to a human who is expected to deliver treatment. Guidelines for handoffs involving AI should be prioritized to assure they are ready when AI is integrated into hospital care.

In addition the danger of the potential delay between decision and result will likely be most evident when a patient managed by Black Box AI fails to recover. The clinician will want to know if a mistake was made in AI based diagnoses or in the therapeutic choices made or both.

Here the need for real-time and detailed communication is evident. The clinician must know which components of the data set comprised the basis for the AI decision and what part of the decision process might be wrong. The answers to these pivotal questions will not be evident when detailed real time communication is lacking and the physician is blind to processing. Again this shows the need for refocused education since transparency and disclosure of RTP detected by the AI are only useful to the physician if she is trained to interpret them.

Fortunately there is already a major body of literature directed toward the study of factors which induce error during human to human communication in the patient care environment. Under these guidelines, a physician handing off care to another must provide detailed communication which explains the time patterns or threshold breaches identified in the data, the diagnostic considerations, and the state and rationale for various therapies applied or under consideration. With those lessons learned, it follows that an AI system which is entrusted with critical clinical decisions, should be considered a human equivalent for the purpose of defining AI communication protocols and standards.

#### **Artificial intelligence as a cause of clinical dependency**

The second problem which may develop over time in the acute care environment comprises intellectual dependency on AI, especially AI which lacks detailed communication capability. The lack of detailed communication renders the state presented in Fig. 2 which will reduce the perceived need for nurses and physicians to learn complex pathophysiology or to intellectually engage the complexity of care. As the perceived need to learn complex pathophysiology diminishes, so too does the competency of the physician or nurse.

Perhaps there will come a time when AI is so highly effective that loss of the quality of the oversight function of nurses and physicians will not be a concern. However, over the next few decades we must focus on the transition state. This is the present state of automated driving, where the performance of AI must still be evaluated and supervised by a human in real time.

### **The need for a new focus of medical education**

There is little doubt that there will be a diminishing role for the portions of traditional medical education which are focused on simple rules and threshold decision making. The simple threshold rules most students have been taught, for example the threshold criteria used to diagnose sepsis, will be abandoned because they are based on data fragments and the AI does not need to keep it simple. Also, physicians will not be able to rely on those simple threshold rules to over read the outputs of the AI because the weakness of those rules is one of the reasons AI is being introduced.

With the emergence of AI, simple threshold based diagnostic and treatment rules will be replaced by AI based protocols which will be much more complex and will include RTP analysis. Fortunately the teaching of threshold rules, which became popular in the late twentieth century, was always an oversimplification so the abandonment of this aspect of medical education will not be much of a loss.

However, if proper steps are not taken soon, the effect of AI on medical education has the potential to cause a much deeper cut. Indeed, it has been suggested that, in the age of AI, physicians will not need to be as intelligent as is required today. Instead, it is argued, the focus of education will be on empathy, comfort, counseling, and end of life care.

Nascent resignation to intellectual subservience to AI is driven, at least in part, by the complexity of the processing performed by the computer. The decision processing of the AI is not easy to teach and it will not be possible for many medical professionals to determine, at the bedside, the integrity of the processing itself. However that does not mean that the loss of the oversight role for the physician and nurse is inevitable. For healthcare workers to maintain their pivotal oversight role they must prepare themselves to cognitively engage the same data, the RTP and the same relationships which are detected and processed by the AI. Just as importantly they must demand that AI for acute care is configured to provide real-time communication of the analysis performed in a format which can be over read by clinicians at the bedside so that physicians can protect their patients from treatment failure due to HTE and the AI and physicians can learn together. If medical educators and physicians do not go forward with both of these steps, they

will become intellectually subservient because they will not be able to oversee the integrity of decisions made by the AI.

The over reading function of the physician or nurse will be facilitated by AI driven cognitive support which presents the detected RTP and the data from which the patterns were detected to the user in an organized way. When this cognitive support is provided, the over reading of such patterns is well within the capability of physicians and nurses trained to understand them but this will require agreement on terminology.

### **Time-series patterns, the “integers of physiology”**

One way to advance ones understanding of biologic time patterns is to start with the most basic pattern pieces, the pattern building blocks of the biological data. Note, we are starting with pattern building blocks of the biologic data not with the biologic data itself. In other words we are going to consider fundamental time patterns of data which are reliably present regardless of the biologic process being considered. One might call these basic time patterns the “integers of physiology” as they are quite simple and universally present in all biologic time series.

Using the basic integers to build a complex pattern means that the complex patterns can be disassembled (factored) into its component parts including the components which define the onset of the pattern. This assures that the variations of complex patterns can be understood, quantified in reliably definable terms.

To understand, and communicate with, and over read time pattern based outputs of advanced medical AI, it is important to begin with a common terminology describing these fundamental integers of physiology. After this we will see how physiologic and pathophysiologic time patterns fit together as we build outward from there.

Fortunately there are only 5 primary fundamental time pattern types. This makes the terminology and the patterns easy to learn. Of course this is only one proposed terminology as no standards have been developed.

These are:

1. 1.

**Perturbation-** (a rise or fall away from the phenotypic or baseline range)

2. 2.

**Recovery** (a rise or fall from a perturbation back toward baseline)

3. 3.

**Reciprocation** (a perturbation followed by its recovery)

4. 4.

**Distortion** (a combination of perturbations induced by a common force such as a drug or invading organism)

5. 5.

**Recovery from a Distortion** (a combination of recoveries from the perturbations which comprise the distortion)

Using images of time patterns as the fundamental communication tool between AI and humans allows both the AI and the physician to succinctly communicate complexity while speaking a common language. Both still having the ability to factor these complex images into their component parts for deeper analysis to learn together and over read each other's work.

### **Biologic time patterns are relational time patterns**

Unless only one dataset from a single point in time is available (which is almost never the case), the patterns detected by the AI are relational time patterns (RTP) of sequential lab values, vitals, pharmaceutical doses, and test results in relation to other things such as image results, other diagnoses, sex, age, etc. Relational time patterns (RTP) are the next level up (from simple time patterns) of pattern complexity. RTP are comprised of two or more time patterns occurring in

temporal relation to each other. Since RTP are built from time pattern “integers” their composition can be widely varied and quite complex depending on how many time patterns are included and of what type. The number of potential time pattern combinations (and therefore RTP) is profound. One example of a complex RTP common in a phenotype of sepsis is comprised of; a fall in absolute neutrophil count, a rise in bands, a fall in platelets, a fall in bicarbonate and a rise in lactate. Together they comprise part of the image of a distortion induced by sepsis.

The acceptance of the view that thresholds can be remembered but RTP are too complex, to numerous and too varied in type to remember would lead to rapid decline in the role of physicians in the management of complex patients and loss of protection provided by physician oversight. There is no alternative to learning these patterns short of giving up the rod of Asclepius to the machines. However, here the computer can assist by presenting the time data in a unified picture so that the patterns are identifiable by the trained eye in the picture rendering the process of relational time pattern detection by physicians and associated AI oversight much easier.

### **Seeing the data of a patient as a factorable time matrix**

To understand any machine (including a biologic one), and to detect its failures, one must first construct the normal relationships of its fundamental parts. The fundamental parts of a human are not its appendages or organs but rather humans are comprised of a massive set of biologic particle densities. Examples of biologic particles include ions, cells, cell fragments, and molecules. The venous platelet count, potassium concentration, an antibody concentration, and the SPO<sub>2</sub> or etCO<sub>2</sub> are all examples of biologic particle densities or measurement surrogates for those densities. Like the parts of a machine, these densities also are held in a highly organized baseline operational state and change over time in a relational way within that baseline operational state.

A human, viewed as a machine, is actually an aggregation of biologic particle densities which are held, in health, in the baseline operational state over time (Fig. 3). As with the machine, one way to cognitively engage a unified patient dataset is by defining a human time matrix model as

a unified aggregation of the fundamental time patterns. With this approach it is possible to perceive an entire patient dataset as a single image which grows over the patient's life time.

**Fig. 3**

A Healthy Human Time Matrix (HTM). Humans are comprised of a matrix of biologic particle densities (e.g. bicarb, sodium, neutrophils), and forces which project along a time axis in a dynamically relational configuration. As represented in the time domain, the HTM is comprised of individual time series (waves) in specific relation to the phenotypic range and to each other (events, forces, lab values, vitals, drugs, etc.)

[Full size image](#)

When a sufficient new force, such as a medication or invading organism, acts on the matrix, the force often causes at least one “perturbation” generated away from the phenotypic range. When the force is removed or a countermeasure is applied, a “recovery” generally occurs in the opposite direction of the perturbation. Perturbations and recoveries are vectors with features of duration, slope, magnitude, acceleration, percent change, etc. The combination of the perturbation and its recovery is a reciprocation and these can be complete (with a full recovery) or incomplete when there is recovery failure.

When a grouping of perturbations caused by a force occurs, this can be seen as a “distortion of the matrix” (Fig. 4). Distortions may have the temporal characteristics of onset, worsening and recovery. A distortions can be factored into its component parts and this factoring can be used by AI or the physician to track back to the earliest perturbation of the distortion when, for example, seeking the force which caused the distortion.

**Fig. 4**

A Distortion of the Human Time Matrix. When an internal or external distortion force (e.g. infection, a drug, a pathogenic molecule) is applied to the matrix, a set of perturbations occur producing a “distortion” of the HTM

[Full size image](#)

Figure 4 illustrates that the distortion force (e.g. infection, a drug, a pathogenic molecule) precedes the individual perturbations which form the distortion. Some perturbations occur early after the onset of the force and some occur late. The types, timing and relational patterns of these perturbations defines the time dimensioned conformation of the distortion.

Distortions produced by different adverse conditions have different conformations which change over time as severity worsens or recovery evolves. Complex conditions such as sepsis may be associated with several phenotypes of distortions.

One way to think about different phenotypes of distortions of a single condition, such as sepsis, is to consider as analogous, sequential images of different groups (image phenotypes) of pneumonia. One image group is that of lobar pneumonia, another bronchopneumonia and a third would be the scattered infiltrates and cavitation often associated with MRSA pneumonia. Although all are pneumonia the images from one group may not be very similar to the images of another and the time patterns of worsening may also be quite dissimilar. Yet, *within* each of these groups the images generally have quite similar time pattern characteristics. The same is true of groups (phenotypes) of distortions of the time matrix for a complex clinical condition.

While the oversight of AI may sound complex, it is not if the data are properly formatted and presented. By modifying medical education and turning the education focus away from simple threshold rules to the recognition of the RTP which AI systems will be detecting, physicians and nurses will be able to continue their pivotal role as overseers of care in the age of AI.



### **Summary of considerations**

Present acute care protocols based by twentieth century threshold decision making are inadequate and AI integration into acute care offers the potential to improve care. Yet, there is a risk that the present role of physicians and nurses as the primary arbiters of acute care in hospitals may be overtaken by computers. While many argue that this transition is inevitable, the process of developing a formal plan to prevent the need to pass control of patient care in the acute care environment to computers should not be further delayed.

The addition of AI as a new bedside intellectual source brings new communication challenges to an environment already associated with a high error rate. When a patient managed under the decision process of AI worsens, at some point the AI will need to “handoff” the patient to a physician. Without transparency this will be a blind handoff. For this reason, any AI system which determines clinical care, must be programmed to provide effective and detailed communication with clinicians as any action which moves care from one caregiver to another is a handoff. This is true even if a computer is the diagnostician and is now handing off the care to a human who is expected to deliver treatment. The patient safety requirements for handoffs, which include full transparency, must be maintained.

Physicians and nurses must control the transition to AI integrated acute care because there is significant risk during the transition period and much of this risk is subtle, unique to the medical environment, and outside the expertise of AI designers.

AI systems detect and analyze relational time patterns and oversight of AI using conventional threshold decision making will not be effective. For this reason, medical education will have to change to provide healthcare workers with the ability to understand AI communications. An increased focus on teaching clinical time pattern recognition is required. Students should be taught in early phases of their medical education to factor complex relational time patterns and to think in relational time pattern terms.

Effective communication between human and artificial intelligence requires a new relational time pattern centered knowledge base and terminology. This terminology must support detailed

and nuanced communication with AI. Experts in safety focused human to human communication in hospitals should lead during this transition process.

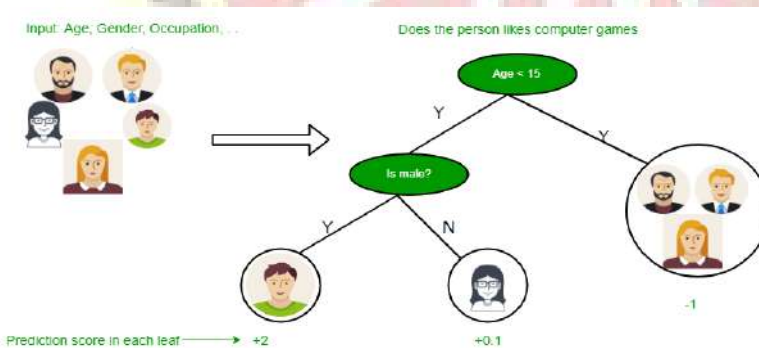
It is the functional hybridization of human and artificial intelligence at the bedside, which offers the greatest hope for safely revolutionizing medical care over the next decade. To achieve this lofty goal early collaboration by experts from many diverse fields of study is required..

## UNIT IV

### Learning From Observation

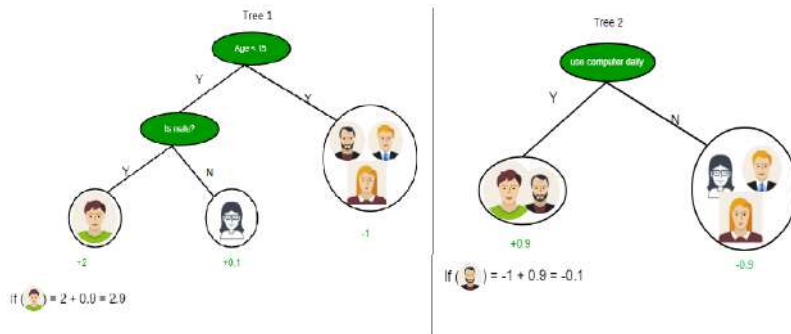
#### Decision Tree Introduction with example

- Decision tree algorithm falls under the category of the supervised learning. They can be used to solve both regression and classification problems.
- Decision tree uses the tree representation to solve the problem in which each leaf node corresponds to a class label and attributes are represented on the internal node of the tree.
- We can represent any boolean function on discrete attributes using the decision tree.



#### Below are some assumptions that we made while using decision tree:

- At the beginning, we consider the whole training set as the root.
- Feature values are preferred to be categorical. If the values are continuous then they are discretized prior to building the model.
- On the basis of attribute values records are distributed recursively.
- We use statistical methods for ordering attributes as root or the internal node.



As you can see from the above image that Decision Tree works on the Sum of Product form which is also known as *Disjunctive Normal Form*. In the above image we are predicting the use of computer in daily life of the people.

In Decision Tree the major challenge is to identification of the attribute for the root node in each level. This process is known as attribute selection. We have two popular attribute selection measures:

1. Information Gain
2. Gini Index

### 1. Information Gain

When we use a node in a decision tree to partition the training instances into smaller subsets the entropy changes. Information gain is a measure of this change in entropy.

**Definition:** Suppose  $S$  is a set of instances,  $A$  is an attribute,  $S_v$  is the subset of  $S$  with  $A = v$ , and  $Values(A)$  is the set of all possible values of  $A$ , then

$$Gain(S, A) = Entropy(S) - \sum_{v \in Values(A)} \frac{|S_v|}{|S|} \cdot Entropy(S_v)$$

#### Entropy

Entropy is the measure of uncertainty of a random variable, it characterizes the impurity of an arbitrary collection of examples. The higher the entropy more the information content.

**Definition:** Suppose  $S$  is a set of instances,  $A$  is an attribute,  $S_v$  is the subset of  $S$  with  $A = v$ , and  $Values(A)$  is the set of all possible values of  $A$ , then

$$Gain(S, A) = Entropy(S) - \sum_{v \in Values(A)} \frac{|S_v|}{|S|} \cdot Entropy(S_v)$$

Example:

For the set  $X = \{a,a,a,b,b,b,b\}$

Total instances: 8

Instances of b: 5

Instances of a: 3

$$\begin{aligned} \text{Entropy } H(X) &= - \left[ \left(\frac{3}{8}\right) \log_2 \frac{3}{8} + \left(\frac{5}{8}\right) \log_2 \frac{5}{8} \right] \\ &= -[0.375 * (-1.415) + 0.625 * (-0.678)] \\ &= -(-0.53 - 0.424) \\ &= 0.954 \end{aligned}$$

**Building Decision Tree using Information Gain**

**The essentials:**

- Start with all training instances associated with the root node
- Use info gain to choose which attribute to label each node with
- Note: No root-to-leaf path should contain the same discrete attribute twice
- Recursively construct each subtree on the subset of training instances that would be classified down that path in the tree.

**The border cases:**

- If all positive or all negative training instances remain, label that node “yes” or “no” accordingly
- If no attributes remain, label with a majority vote of training instances left at that node
- If no instances remain, label with a majority vote of the parent’s training instances

**Example:**

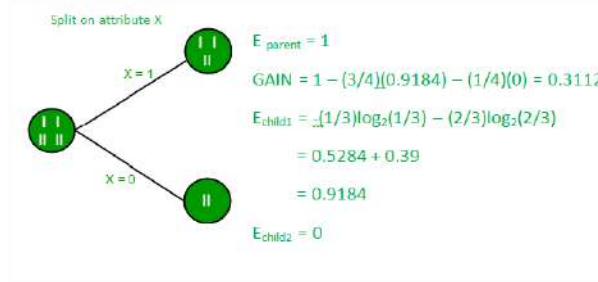
Now, let's draw a Decision Tree for the following data using Information gain.

**Training set: 3 features and 2 classes**

X	Y	Z	C
1	1	1	I
1	1	0	I

$X$	$Y$	$Z$	$C$
0	0	1	II
1	0	0	II

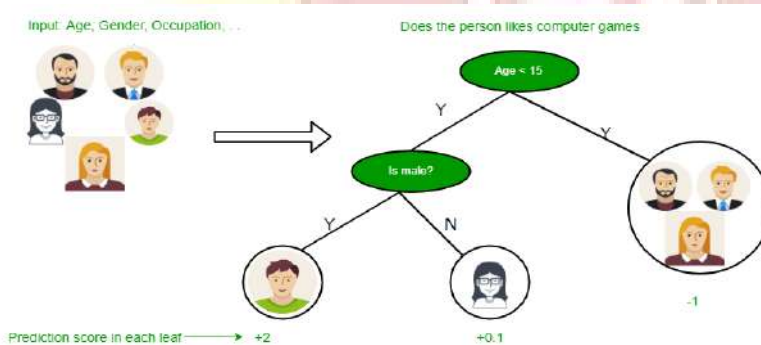
Here, we have 3 features and 2 output classes. To build a decision tree using Information gain. We will take each of the feature and calculate the information for each feature.



**Split on feature X**

### Decision Tree Introduction with example

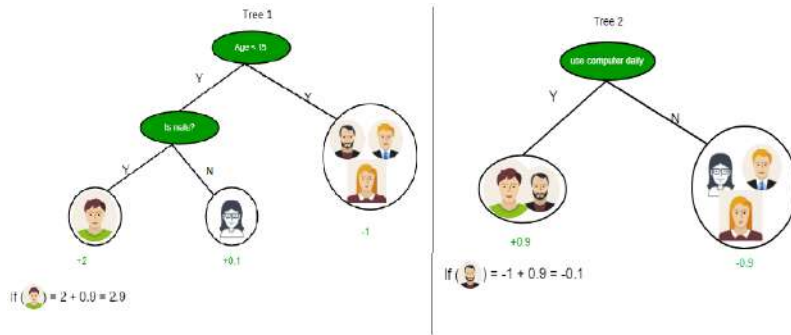
- Decision tree algorithm falls under the category of the supervised learning. They can be used to solve both regression and classification problems.
- Decision tree uses the tree representation to solve the problem in which each leaf node corresponds to a class label and attributes are represented on the internal node of the tree.
- We can represent any boolean function on discrete attributes using the decision tree.



**Below are some assumptions that we made while using decision tree:**

- At the beginning, we consider the whole training set as the root.
- Feature values are preferred to be categorical. If the values are continuous then they are discretized prior to building the model.
- On the basis of attribute values records are distributed recursively.

- We use statistical methods for ordering attributes as root or the internal node.



As you can see from the above image that Decision Tree works on the Sum of Product form which is also known as *Disjunctive Normal Form*. In the above image we are predicting the use of computer in daily life of the people.

In Decision Tree the major challenge is to identification of the attribute for the root node in each level. This process is known as attribute selection. We have two popular attribute selection measures:

Information Gain

## 1. Gini Index

### 1. Information Gain

When we use a node in a decision tree to partition the training instances into smaller subsets the entropy changes. Information gain is a measure of this change in entropy.

**Definition:** Suppose  $S$  is a set of instances,  $A$  is an attribute,  $S_v$  is the subset of  $S$  with  $A = v$ , and  $Values(A)$  is the set of all possible values of  $A$ , then

$$Gain(S, A) = Entropy(S) - \sum_{v \in Values(A)} \frac{|S_v|}{|S|} \cdot Entropy(S_v)$$

### Entropy

Entropy is the measure of uncertainty of a random variable, it characterizes the impurity of an arbitrary collection of examples. The higher the entropy more the information content.

**Definition:** Suppose  $S$  is a set of instances,  $A$  is an attribute,  $S_v$  is the subset of  $S$  with  $A = v$ , and  $Values(A)$  is the set of all possible values of  $A$ , then

$$Gain(S, A) = Entropy(S) - \sum_{v \in Values(A)} \frac{|S_v|}{|S|} \cdot Entropy(S_v)$$

*Example:*

For the set  $X = \{a,a,a,b,b,b,b,b\}$

Total instances: 8

Instances of b: 5

Instances of a: 3

$$\begin{aligned} \text{Entropy } H(X) &= - \left[ \left(\frac{3}{8}\right) \log_2 \frac{3}{8} + \left(\frac{5}{8}\right) \log_2 \frac{5}{8} \right] \\ &= -[0.375 * (-1.415) + 0.625 * (-0.678)] \\ &= -(-0.53 - 0.424) \\ &= 0.954 \end{aligned}$$

### **Building Decision Tree using Information Gain**

#### **The essentials:**

- Start with all training instances associated with the root node
- Use info gain to choose which attribute to label each node with
- Note: No root-to-leaf path should contain the same discrete attribute twice
- Recursively construct each subtree on the subset of training instances that would be classified down that path in the tree.

#### **The border cases:**

- If all positive or all negative training instances remain, label that node “yes” or “no” accordingly
- If no attributes remain, label with a majority vote of training instances left at that node
- If no instances remain, label with a majority vote of the parent’s training instances

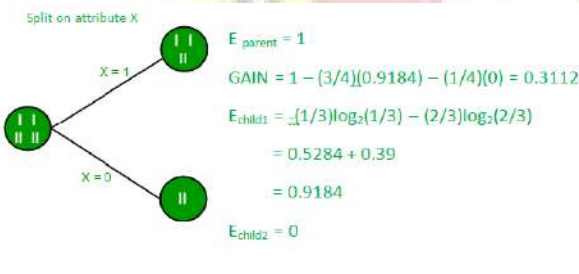
#### **Example:**

Now, let's draw a Decision Tree for the following data using Information gain.

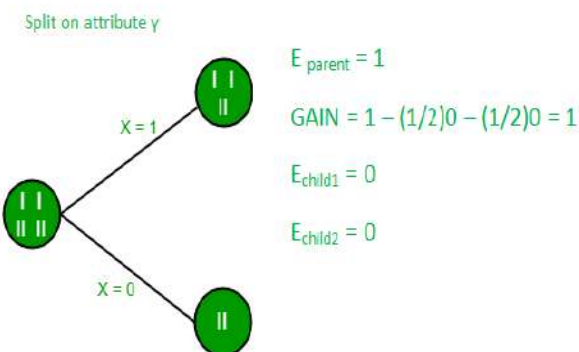
**Training set: 3 features and 2 classes**

X	Y	Z	C
1	1	1	I
1	1	0	I
0	0	1	II
1	0	0	II

Here, we have 3 features and 2 output classes. To build a decision tree using Information gain. We will take each of the feature and calculate the information for each feature.

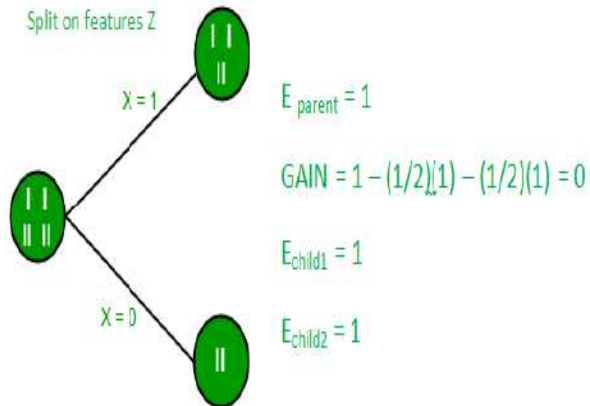


**Split on feature X**



**Split on feature Y**

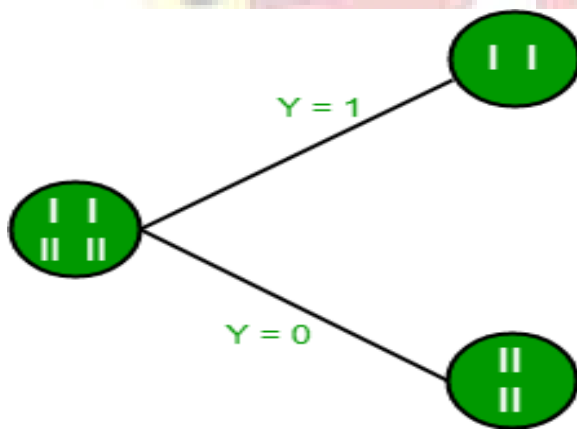




### Split on feature Z

From the above images we can see that the information gain is maximum when we make a split on feature Y. So, for the root node best suited feature is feature Y. Now we can see that while splitting the dataset by feature Y, the child contains pure subset of the target variable. So we don't need to further split the dataset.

The final tree for the above dataset would be look like this:



## 2. Gini Index

- Gini Index is a metric to measure how often a randomly chosen element would be incorrectly identified.
- It means an attribute with lower Gini index should be preferred.
- Sklearn supports "Gini" criteria for Gini Index and by default, it takes "gini" value.
- The Formula for the calculation of the of the Gini Index is given below.

$$GiniIndex = 1 - \sum_j p_j^2$$

**Example:**

Lets consider the dataset in the image below and draw a decision tree using gini index.

Index	A	B	C	D	E
1	4.8	3.4	1.9	0.2	positive
2	5	3	1.6	1.2	positive
3	5	3.4	1.6	0.2	positive
4	5.2	3.5	1.5	0.2	positive
5	5.2	3.4	1.4	0.2	positive
6	4.7	3.2	1.6	0.2	positive
7	4.8	3.1	1.6	0.2	positive
8	5.4	3.4	1.5	0.4	positive
9	7	3.2	4.7	1.4	negative
10	6.4	3.2	4.7	1.5	negative
11	6.9	3.1	4.9	1.5	negative
12	5.5	2.3	4	1.3	negative
13	6.5	2.8	4.6	1.5	negative
14	5.7	2.8	4.5	1.3	negative
15	6.3	3.3	4.7	1.6	negative
16	4.9	2.4	3.3	1	negative

In the dataset above there are 5 attributes from which attribute E is the predicting feature which contains 2(Positive & Negative) classes. We have equal proportion for both the classes. In Gini Index, we have to choose some random values to categorize each attribute. These values for this dataset are:

A	B	C	D
$\geq 5$	$\geq 3.0$	$\geq 4.2$	$\geq 1.4$
$< 5$	$< 3.0$	$< 4.2$	$< 1.4$

Calculating Value	Gini Index	for	Var	A:
	$\geq$	5:		12
Attribute A	$\geq$	5	& class =	positive: $\frac{5}{12}$
Attribute A	$\geq$	5	& class =	negative: $\frac{7}{12}$
Gini(5, 7)	=	1	-	$\left[ \left(\frac{5}{12}\right)^2 + \left(\frac{7}{12}\right)^2 \right] = 0.4860$
	$<$	5:		4
Attribute A	$<$	5	& class =	positive: $\frac{3}{4}$
Attribute A	$<$	5	& class =	negative: $\frac{1}{4}$
Gini(3, 1)	=	1	-	$\left[ \left(\frac{3}{4}\right)^2 + \left(\frac{1}{4}\right)^2 \right] = 0.375$

By adding weight and sum each of the gini indices:

$$gini(Target, A) = \left(\frac{12}{16}\right) * (0.486) + \left(\frac{4}{16}\right) * (0.375) = 0.45825$$

Calculating Value	Gini Index	for	Var	B:
	$\geq$	3:		12
Attribute B	$\geq$	3	& class =	positive: $\frac{8}{12}$
Attribute B	$\geq$	5	& class =	negative: $\frac{4}{12}$
Gini(5, 7)	=	1	-	$\left[ \left(\frac{8}{12}\right)^2 + \left(\frac{4}{12}\right)^2 \right] = 0.4460$
	$<$	3:		4
Attribute A	$<$	3	& class =	positive: $\frac{0}{4}$

Attribute A < 3 & class = negative:  $\frac{4}{4}$

$$Gini(3, 1) = 1 - \left[ \left(\frac{0}{4}\right)^2 + \left(\frac{4}{4}\right)^2 \right] = 1$$

By adding weight and sum each of the gini indices:

$$gini(Target, B) = \left(\frac{12}{16}\right) * (0.446) + \left(\frac{0}{16}\right) * (0) = 0.3345$$

Using the same approach we can calculate the Gini index for C and D attributes.

Positive Negative

For A |>= 5.0 5 7

|< 5 3 1

Gini Index of A = 0.45825

Positive Negative

For B |>= 3.0 8 4

|< 3.0 0 4

Gini Index of B = 0.3345

Positive Negative

For C |>= 4.2 0 6

|< 4.2 8 2

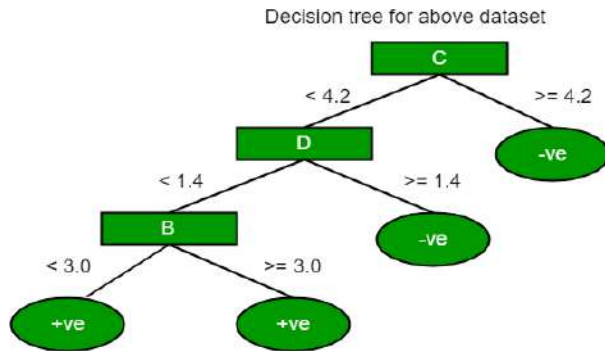
Gini Index of C = 0.2

Positive Negative

For D |>= 1.4 0 5

|< 1.4 8 3

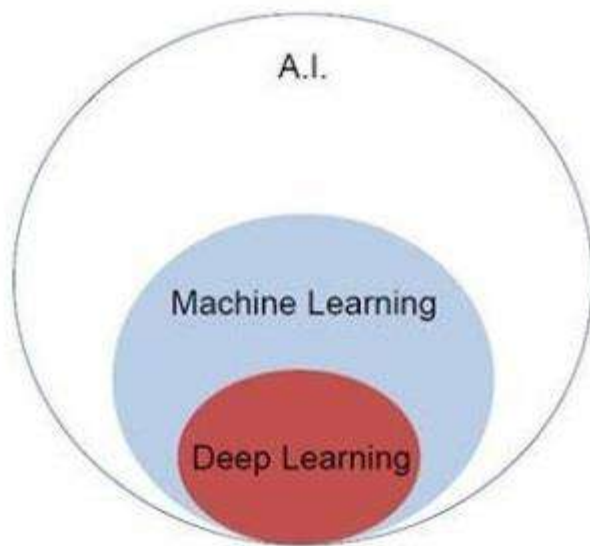
Gini Index of D = 0.273



Artificial Intelligence includes the simulation process of human intelligence by machines and special computer systems. The examples of artificial intelligence include learning, reasoning and self-correction. Applications of AI include speech recognition, expert systems, and image recognition and machine vision.

Machine learning is the branch of artificial intelligence, which deals with systems and algorithms that can learn any new data and data patterns.

Let us focus on the Venn diagram mentioned below for understanding machine learning and deep learning concepts.



Machine learning includes a section of machine learning and deep learning is a part of machine learning. The ability of program which follows machine learning concepts is to improve its performance of observed data. The main motive of data transformation is to improve its

knowledge in order to achieve better results in the future, provide output closer to the desired output for that particular system. Machine learning includes “pattern recognition” which includes the ability to recognize the patterns in data.

The patterns should be trained to show the output in desirable manner.

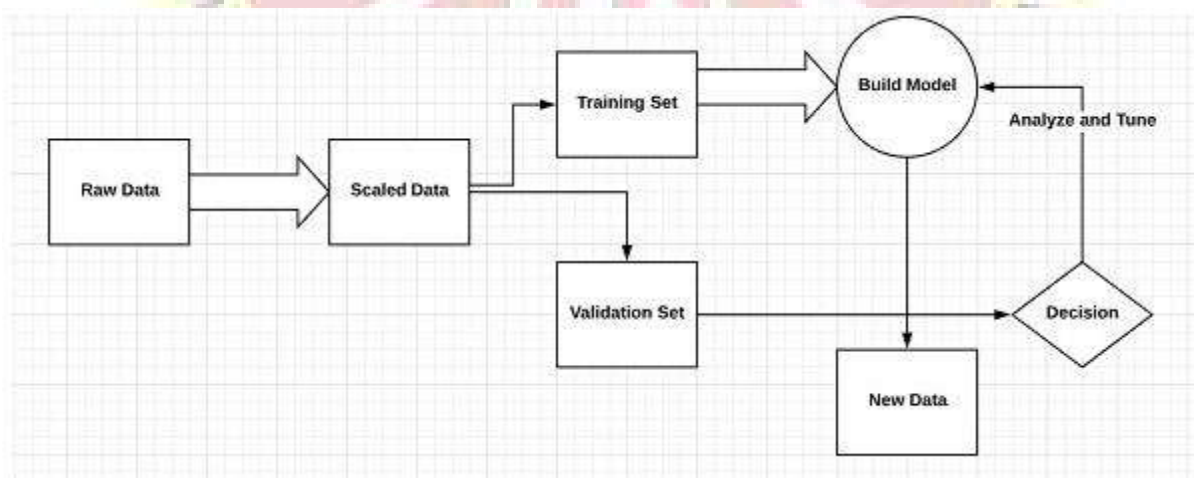
Machine learning can be trained in two different ways –

- Supervised training
- Unsupervised training

### Supervised Learning

Supervised learning or supervised training includes a procedure where the training set is given as input to the system wherein, each example is labeled with a desired output value. The training in this type is performed using minimization of a particular loss function, which represents the output error with respect to the desired output system.

After completion of training, the accuracy of each model is measured with respect to disjoint examples from training set, also called the validation set.



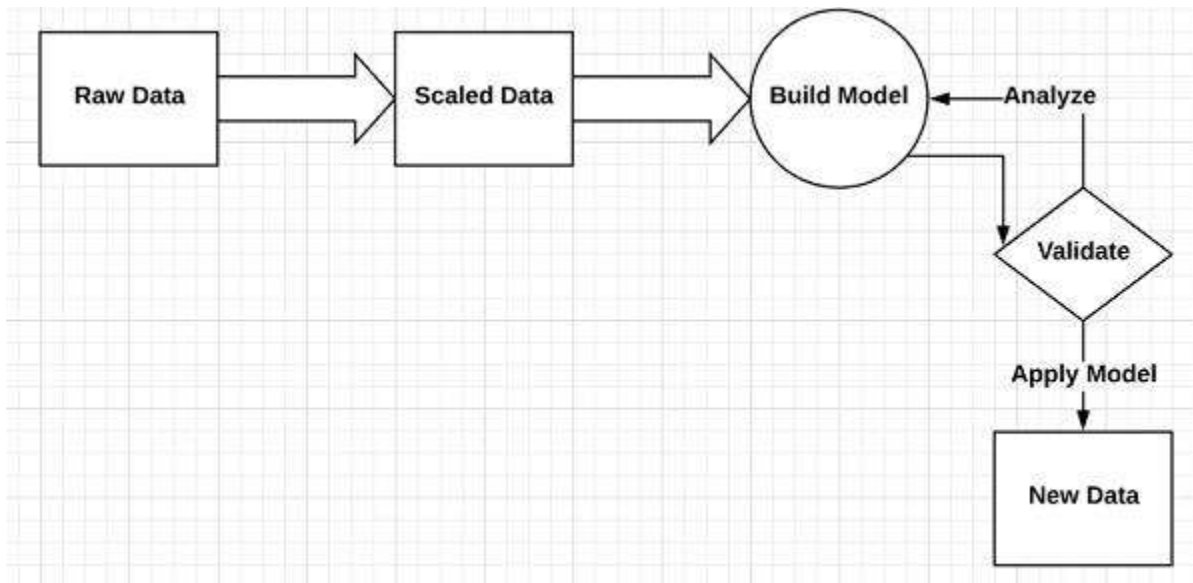
The best example to illustrate “Supervised learning” is with a bunch of photos given with information included in them. Here, the user can train a model to recognize new photos.

### Unsupervised Learning

In unsupervised learning or unsupervised training, include training examples, which are not labeled by the system to which class they belong. The system looks for the data, which share

common characteristics, and changes them based on internal knowledge features. This type of learning algorithms are basically used in clustering problems.

The best example to illustrate “Unsupervised learning” is with a bunch of photos with no information included and user trains model with classification and clustering. This type of training algorithm works with assumptions as no information is given.

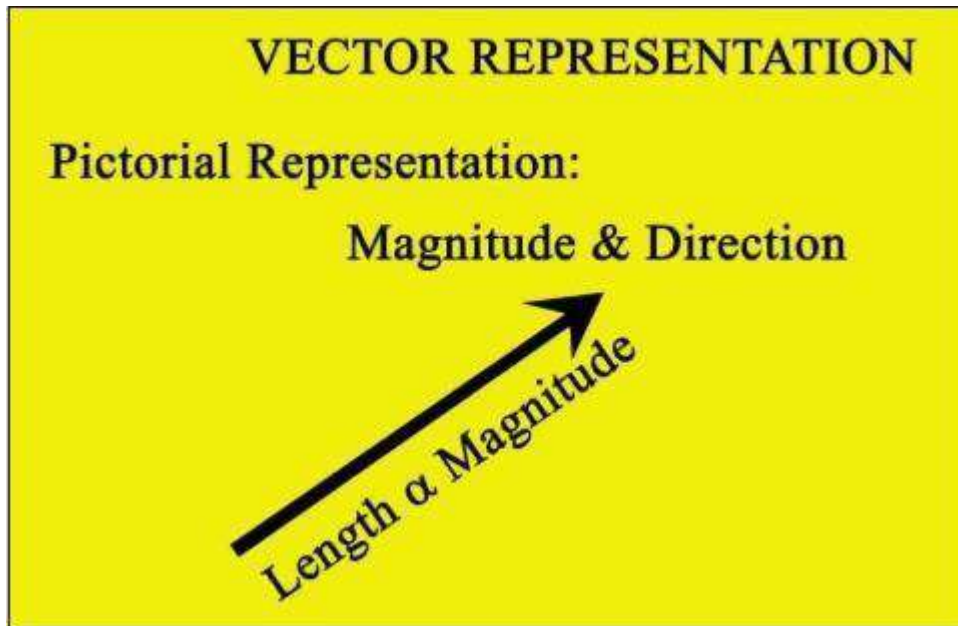


It is important to understand mathematical concepts needed for TensorFlow before creating the basic application in TensorFlow. Mathematics is considered as the heart of any machine learning algorithm. It is with the help of core concepts of Mathematics, a solution for specific machine learning algorithm is defined.

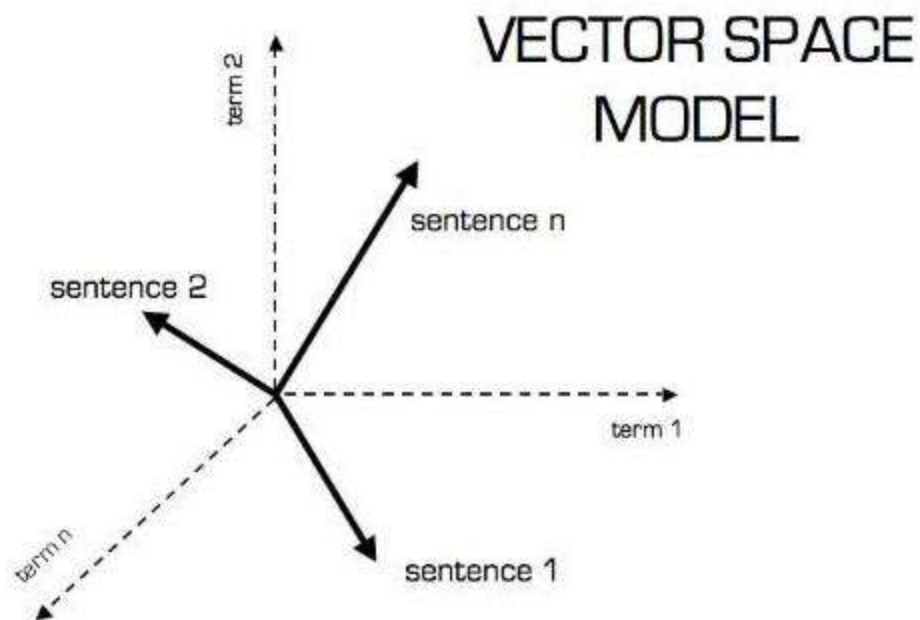
### **Vector**

An array of numbers, which is either continuous or discrete, is defined as a vector. Machine learning algorithms deal with fixed length vectors for better output generation.

Machine learning algorithms deal with multidimensional data so vectors play a crucial role.



The pictorial representation of vector model is as shown below –



### Scalar

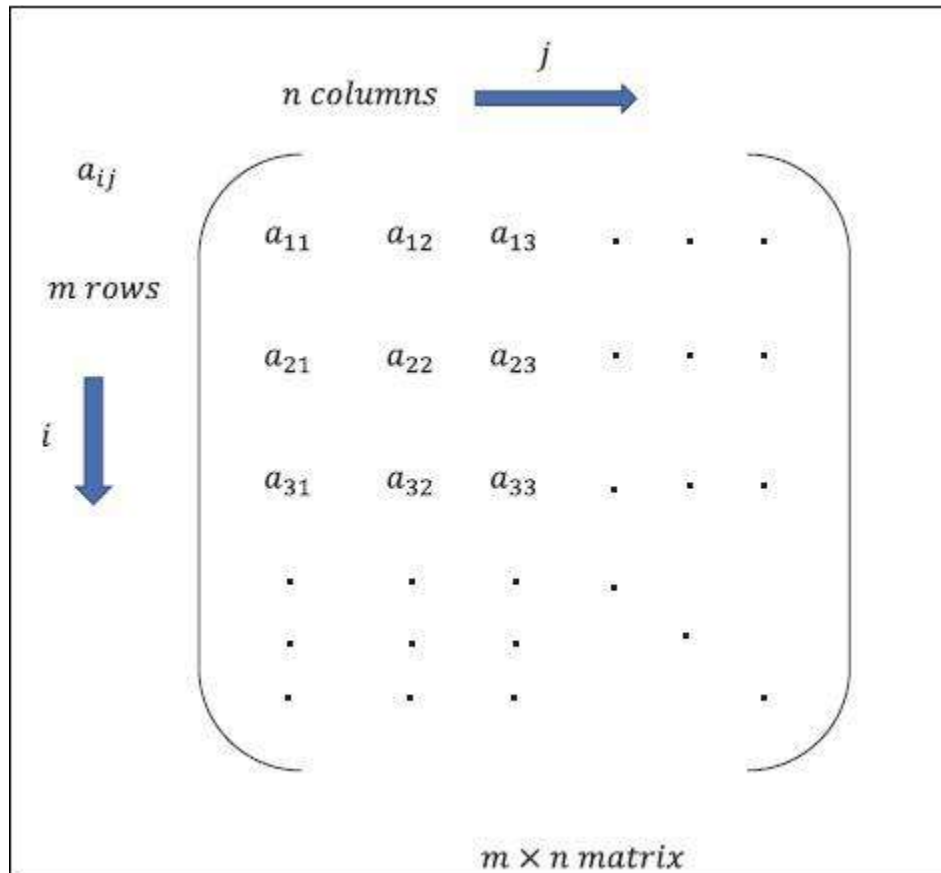
Scalar can be defined as one-dimensional vector. Scalars are those, which include only magnitude and no direction. With scalars, we are only concerned with the magnitude.

Examples of scalar include weight and height parameters of children.



## Matrix

Matrix can be defined as multi-dimensional arrays, which are arranged in the format of rows and columns. The size of matrix is defined by row length and column length. Following figure shows the representation of any specified matrix.



Consider the matrix with “m” rows and “n” columns as mentioned above, the matrix representation will be specified as “m\*n matrix” which defined the length of matrix as well.

## Mathematical Computations

In this section, we will learn about the different Mathematical Computations in TensorFlow.

### Addition of matrices

Addition of two or more matrices is possible if the matrices are of the same dimension. The addition implies addition of each element as per the given position.

Consider the following example to understand how addition of matrices works –

Example:  $A=[1324]B=[5768]$  then  $A+B=[1+53+72+64+8]=[610812]$  Example:  $A=[1234]B=[5678]$  then  $A+B=[1+52+63+74+8]=[681012]$

### Subtraction of matrices

The subtraction of matrices operates in similar fashion like the addition of two matrices. The user can subtract two matrices provided the dimensions are equal.

Example:  $A=[1324]B=[5768]$  then  $A-B=[1-53-72-64-8]=[-4-4-4-4]$  Example:  $A=[1234]B=[5678]$  then  $A-B=[1-52-63-74-8]=[-4-4-4-4]$

### Multiplication of matrices

For two matrices  $A$   $m \times n$  and  $B$   $p \times q$  to be multipliable,  $n$  should be equal to  $p$ . The resulting matrix is –

$C$   $m \times q$

$A=[1324]B=[5768]A=[1234]B=[5678]$

$c_{11}=[12][57]=1 \times 5 + 2 \times 7 = 19$   $c_{12}=[12][68]=1 \times 6 + 2 \times 8 = 22$   $c_{11}=[12][57]=1 \times 5 + 2 \times 7 = 19$   $c_{12}=[12][68]=1 \times 6 + 2 \times 8 = 22$

$c_{21}=[34][57]=3 \times 5 + 4 \times 7 = 43$   $c_{22}=[34][68]=3 \times 6 + 4 \times 8 = 50$   $c_{21}=[34][57]=3 \times 5 + 4 \times 7 = 43$   $c_{22}=[34][68]=3 \times 6 + 4 \times 8 = 50$

$C=[c_{11}c_{21}c_{12}c_{22}]=[19432250]$   $C=[c_{11}c_{12}c_{21}c_{22}]=[19224350]$

### Transpose of matrix

The transpose of a matrix  $A$ ,  $m \times n$  is generally represented by  $A^T$  (transpose)  $n \times m$  and is obtained by transposing the column vectors as row vectors.

Example:  $A=[1324]$  then  $A^T=[1234]$  Example:  $A=[1234]$  then  $A^T=[1324]$

### Dot product of vectors

Any vector of dimension  $n$  can be represented as a matrix  $v = R^n \times 1$ .

$$v_1 = [v_{11} \ v_{12} \ \dots \ v_{1n}] \quad v_2 = [v_{21} \ v_{22} \ \dots \ v_{2n}] \quad v_1 \cdot v_2 = [v_{11}v_{21} \ v_{12}v_{22} \ \dots \ v_{1n}v_{2n}]$$

The dot product of two vectors is the sum of the product of corresponding components – Components along the same dimension and can be expressed as

$$v_1 \cdot v_2 = v_1^T v_2 = v_2^T v_1 = v_{11}v_{21} + v_{12}v_{22} + \dots + v_{1n}v_{2n} = \sum_{k=1}^n v_{1k}v_{2k} \quad v_1 \cdot v_2 = v_1^T v_2 = v_2^T v_1 = v_{11}v_{21} + v_{12}v_{22} + \dots + v_{1n}v_{2n} = \sum_{k=1}^n v_{1k}v_{2k}$$

The example of dot product of vectors is mentioned below –

$$\text{Example: } v_1 = [1 \ 2 \ 3] \quad v_2 = [3 \ 5 \ -1] \quad v_1 \cdot v_2 = v_1^T v_2 = 1 \times 3 + 2 \times 5 - 3 \times 1 = 10$$

### **Statistical Learning Methods**

Statistical Learning is a set of tools for understanding data. These tools broadly come under two classes: supervised learning & unsupervised learning. Generally, supervised learning refers to predicting or estimating an output based on one or more inputs. Unsupervised learning, on the other hand, provides a relationship or finds a pattern within the given data without a supervised output.

#### **What is Statistical Learning?**

Let, suppose that we observe a response  $Y$  and  $p$  different predictors  $X = (X_1, X_2, \dots, X_p)$ . In general, we can say:

$$Y = f(X) + \varepsilon$$

Here  $f$  is an unknown function, and  $\varepsilon$  is the *random error term*.

*In essence, statistical learning refers to a set of approaches for estimating  $f$ .*

In cases where we have set of  $X$  readily available, but the output  $Y$ , not so much, the error averages to zero, and we can say:

$$\mathbb{E} = f(X)$$

where  $\hat{f}$  represents our estimate of  $f$  and  $\hat{Y}$  represents the resulting prediction.

Hence for a set of predictors  $X$ , we can say:

$$E(Y - \hat{Y})^2 = E[f(X) + \epsilon - \hat{f}(X)]^2 \Rightarrow E(Y - \hat{Y})^2 = [f(X) - \hat{f}(X)]^2 + \text{Var}(\epsilon)$$

where,

- $E(Y - \hat{Y})^2$  represents the *expected value* of the squared difference between actual and expected result.
- $[f(X) - \hat{f}(X)]^2$  represents the **reducible error**. It is reducible because we can potentially improve the accuracy of  $\hat{f}$  by better modeling.
- $\text{Var}(\epsilon)$  represents the **irreducible error**. It is irreducible because no matter how well we estimate  $f$ , we cannot reduce the error introduced by *variance* in  $\epsilon$ .

### Regression Vs Classification Problem

Variables,  $Y$ , can be broadly be characterized as *quantitative* or *qualitative* (also known as *categorical*). Quantitative variables take on numerical values, e.g., age, height, income, price, and much more. Estimating quantitative responses is often termed as *a regression problem*. Qualitative variables take on categorical values, e.g., gender, brand, parts of speech, and much more. Estimating qualitative responses is often termed as *a classification problem*.

There is no free lunch in statistics: no one method dominates all other over all possible data sets.

### Variance And Bias

*Variance* refers to the amount by which  $\hat{f}$  would change if we estimated with different training data sets. In general, when we over-fit a model on a given training data set (reducible error in training set is very low but on test set is very high), we get a model that has higher variance since any change in the data points would result in a significantly different model.

*Bias* refers to the error introduced by approximating a real-life problem, which may be extremely complicated by a much simpler model — for example, modeling non-linear problems with a linear model. In general, when we over-fit, a model on given data set it results in very less bias.

This results in the variance bias trade-off.

As we fit the model over a given data set, the bias tends to decrease faster than the variance increases initially. Consequently, the expected test error(reducible) declines. However, at some point, when over-fitting starts, there is a little impact on the bias, but variance starts to increase rapidly when this happens the test error increases.

## Linear Regression

Linear regression is a statistical method belonging to supervised learning used for predicting quantitative responses.

**Simple Linear Regression** approach predicts a quantitative response  $Y$  based on a single variable  $X$  assuming a linear relationship. We can say :

$$Y \approx \beta_0 + \beta_1 X$$

Our job is now to *estimate*  $\beta_0$  and  $\beta_1$ , the parameters/coefficients of our model based on the training data set, such that the hyperplane(in this case a line) is *close* to the training data set. Many criteria can estimate the closeness, the most common being *least square*.

The sum of the square of the difference between all observed response and the predicted response formulates to *Residual Sum Of Squares(RSS)*.

## Problems in Linear Regression

- *Non-linearity of the response-predictor relationships.*
- *Correlation of error terms.*

- *The non-constant variance of error terms.*
- *Outliers:* when the actual prediction is very *far* from the estimated one, can arise due to inaccurate recording of data.
- *High-leverage points:* Unusual values of the predictors impact the regression line known as high leverage points.
- *Collinearity:* where two or more predictor variables are closely related to each other, it may be challenging to weed out the individual effect of a single predictor variable.

### KNN Regression

KNN Regression is a non-parametric approach towards estimating or predicting values, which do not assume the form of  $f(X)$ . It estimates/predicts  $f(x_0)$  where  $x_0$  is a prediction point by averaging out all  $N_0$  responses closest to  $x_0$ . We can say:

$$\hat{f}(x_0) = \frac{1}{K} \sum_{x_i \in N_0} y_i$$

$$f(x_0) = \frac{1}{K} \sum_{x_i \in N_0} y_i$$

*Note: If  $K$  is small, the fit would be flexible and any change in the data would result in a different fit, hence for small  $K$  the variance would be high and bias low; conversely, if  $K$  is large, it might mask some structure in the data hence the bias would be high.*

### The Classification Problem

The responses as we discussed till now, may not always be *quantitative*, it can be also *qualitative*, predicting these qualitative responses is called classification.

We will discuss various statistical approaches to classification including:

- SVM
- *Logistic Regression*
- *KNN Classifier*
- *GAM*
- *Trees*
- *Random Forest*
- *Boosting*

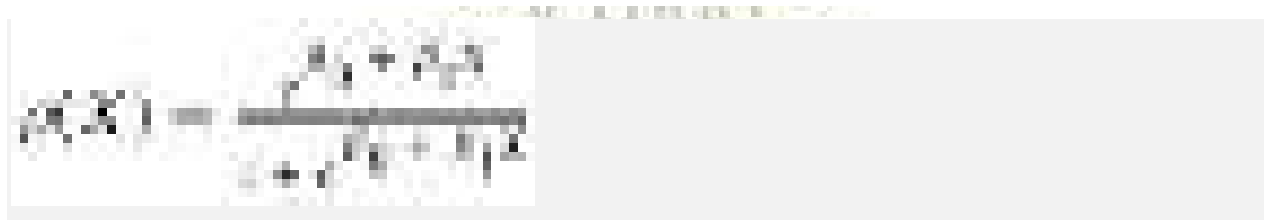
### **Support Vector Machine(SVM)**

SVM or support vector machine is the classifier that maximizes the margin. The goal of a classifier in our example below is to find a line or (n-1) dimension hyper-plane that separates the two classes present in the n-dimensional space. I have written a detailed [article](#) explaining the derivation and formulation of SVM. In my opinion, it is one of the most powerful techniques in our tool box of statistical methods in AI.

Logistic Regression

Logistic model models the probability of output response  $Y$  belonging to a particular category.

We can say:



$$\rho(X) = \frac{e^{\beta_0 + \beta_1 X}}{1 + e^{\beta_0 + \beta_1 X}}$$

Applying componendo dividendo we get:

$$\Rightarrow \frac{\rho(X)}{1 - \rho(X)} = e^{\beta_0 + \beta_1 X}$$

which is nothing but the odds.

$$\Rightarrow \ln\left(\frac{\rho(X)}{1 - \rho(X)}\right) = \beta_0 + \beta_1 X$$

*log - odds / logit*

For estimating the beta coefficients, we can use maximum likelihood. The basic idea is to estimate the betas such that the estimated value and observed value of the results are as close as possible. In a binary classification, with observed classes as 1 and 0, we can say the likelihood function would look like:

$$l(\beta_0, \beta_1) = \prod_{i : y_i = 1} \hat{p}(x_i) \prod_{\tilde{i} : y_{\tilde{i}} = 0} (1 - \hat{p}(x_{\tilde{i}}))$$



## **KNN Classifier**

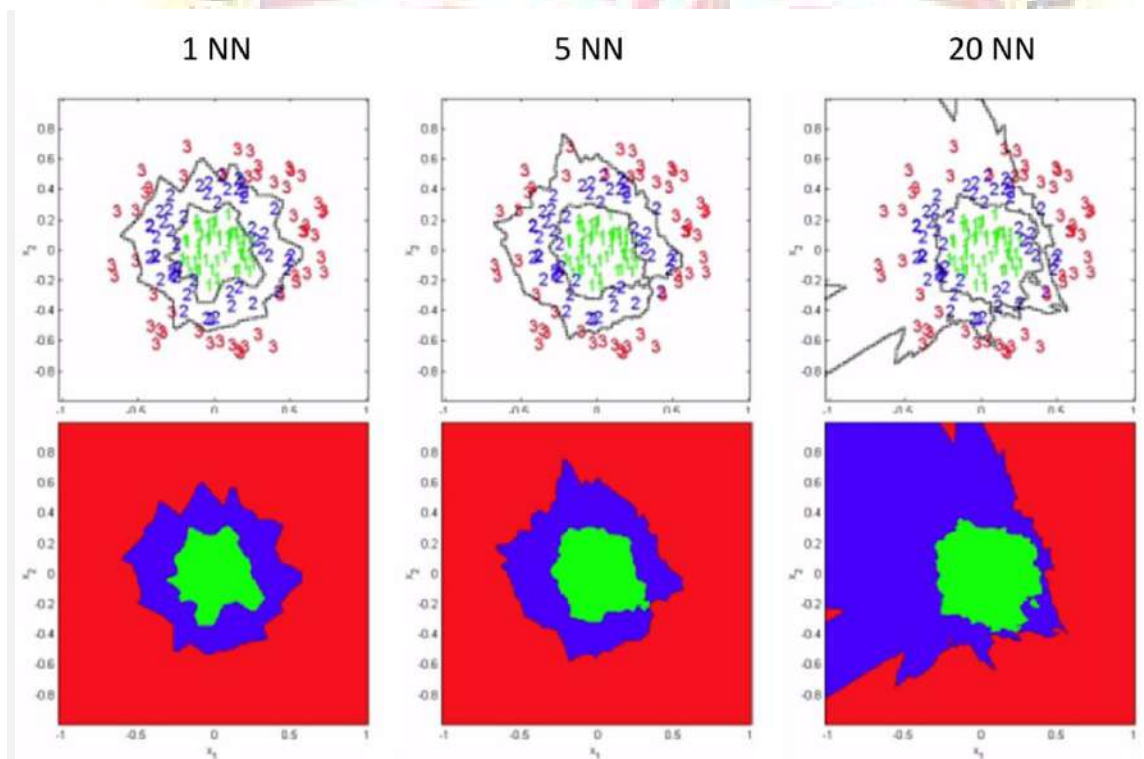
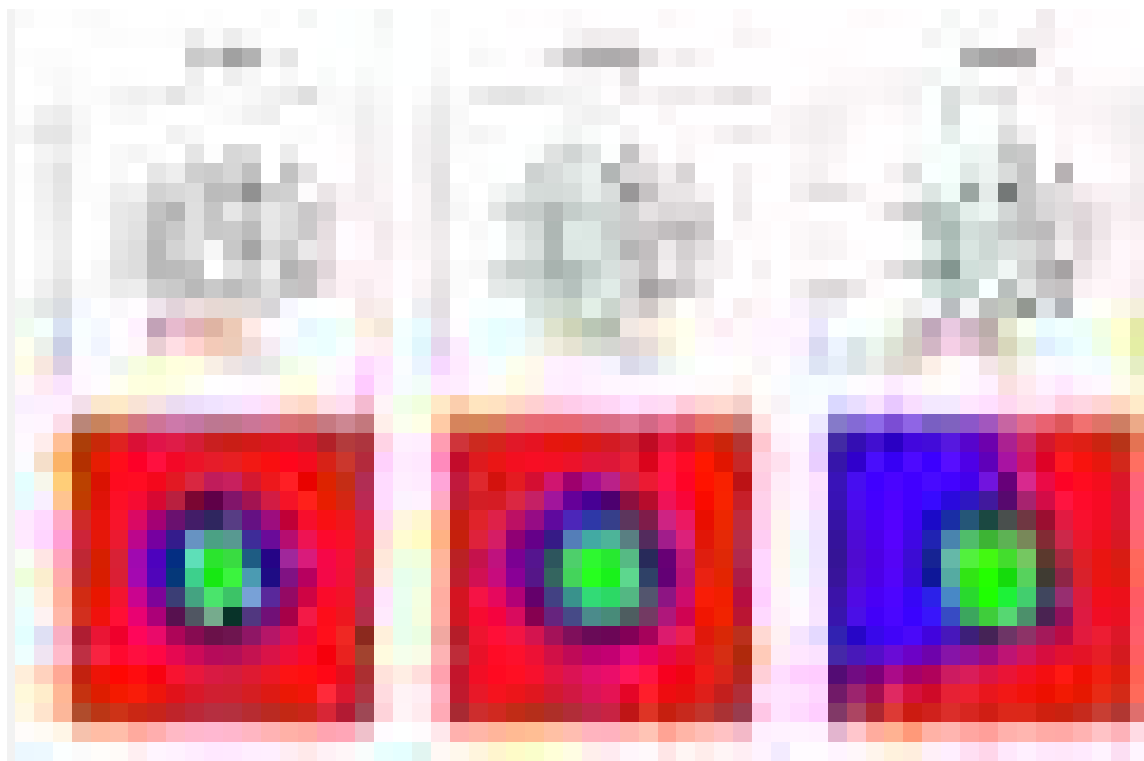
KNN(K nearest neighbors) Classifier is a lazy learning technique, where the training data set is represented on a Euclidean hyperplane, and test data is assigned the labels based on the K Euclidean distance metrics.

### **Practical Aspects**

- K should be chosen empirically and preferably odd to avoid tie situation.
- KNN should have both discrete and continuous target functions.
- Weighted contribution(e.g. distance based) from different neighbors can be used computing the final label.

*Note: Performance of KNN degrades when the data is high dimensional. This can be avoided by providing weights to the features itself.*

### **Effect Of K on the decision boundary**



### Advantages Of KNN

- We can learn a complex target function.

- Zero loss of any information.

### Disadvantages of KNN

- Classification cost of new instances is very high.
- Significant computation takes place at classification time.

### Generalized Additive Models

GAM provides a generalized framework extending standard multivariable linear regression with the nonlinear function of each variable while maintaining its additive nature. Thus, all nonlinear functions can be independently calculated and added later.

*Note: GAM like linear regression can be applied to both quantitative and qualitative responses.*

### Trees, Random Forest, Boosting, and Bagging

Trees or decision trees are useful and straightforward methods for both regression and classification involving segmenting the predictor space into simple regions.

Typically decision trees are drawn *upside down* meaning the leaves are at the bottom of the tree. The points where the predictor space is split are known as *internal nodes*, and the *leaf nodes* or *terminal nodes* are the ones which given the predictions. Segments joining the nodes are known as *branches*.

For prediction, we take a *top-down* (at the first point all the observation belongs to just one region), *greedy* (best split is made in the particular step) approach known as recursive binary fitting.

There are strategies like [tree pruning](#) that solves the over-fitting problem of trees by cutting some of the branches to get a small sub-tree.

For a classification problem, we either use Gini index,

$$G = \sum_{k=1}^K \hat{p}_{mk} (1 - \hat{p}_{mk})$$

or entropy

$$D = - \sum_{k=1}^K \hat{p}_{mk} \log \hat{p}_{mk}$$

to represent the purity of a node, where  $P_{mk}$  is the proportion of samples in the  $m$ th region from  $k$ th class.

Decision trees still suffer from high variance and are not competitive with other supervised approaches. Therefore, we introduce random forest boosting and bagging.

### **Bagging**

Bagging is a general-purpose method to reduce variance in a statistical learning method. The core idea is that averaging a set of observations reduces variance. Hence we do a random sampling of our data multiple times, and for each sample, we construct a tree and average out all the predictions to give a low variance result.

### **Random Forest**

When in the collection of bagged trees, a fix  $k$  predictors are chosen at random from each tree having total  $m$  predictors ( $k < m$ ), then bagging becomes a random forest.

This is done because most of the bagged trees would look more or less the same. Hence, the predictions of individual bag trees would be highly co-related. Therefore, there would not be much reduction in the variance of our inferences. Random forests can be thought of as the process of de-correlating bagged trees.

### **Boosting**

Boosting approach is a slow learning statistical method, where classifiers are learned on modified data set *sequentially*. In the context of decision trees, each tree is grown using information from the previous trees. This way, we do not fit a single large tree.

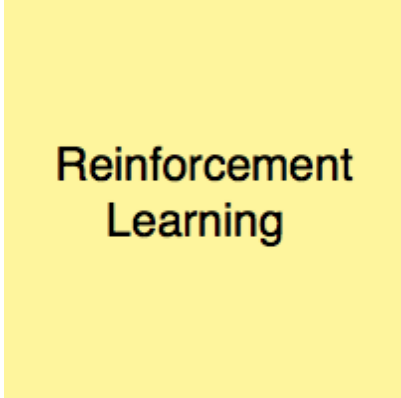
### **Unsupervised Learning**

All the above methods had some form of annotated data set. But when we want to learn patterns in our data without any annotations unsupervised learning comes into the picture.

The most widely used statistical method for unsupervised learning is *K-Means Clustering*. We take k random points in our data set and map all other points to one of the K regions based on their closeness to K chosen random points. Then we change the K random points to the centroid of the clusters thus formed. We do that until we observe a negligible change in the cluster formed after each iteration.

There are other techniques like PCA in unsupervised learning that are used a lot, but for now, we end here.

### **Reinforcement Learning Tutorial**



**Reinforcement  
Learning**

Our Reinforcement learning tutorial will give you a complete overview of reinforcement learning, including MDP and Q-learning. In RL tutorial, you will learn the below topics:

- [What is Reinforcement Learning?](#)
- [Terms used in Reinforcement Learning.](#)
- [Key features of Reinforcement Learning.](#)
- [Elements of Reinforcement Learning.](#)
- [Approaches to implementing Reinforcement Learning.](#)
- [How does Reinforcement Learning Work?](#)
- [The Bellman Equation.](#)
- [Types of Reinforcement Learning.](#)
- [Reinforcement Learning Algorithm.](#)
- [Markov Decision Process.](#)
- [What is Q-Learning?](#)
- [Difference between Supervised Learning and Reinforcement Learning.](#)
- [Applications of Reinforcement Learning.](#)
- [Conclusion.](#)

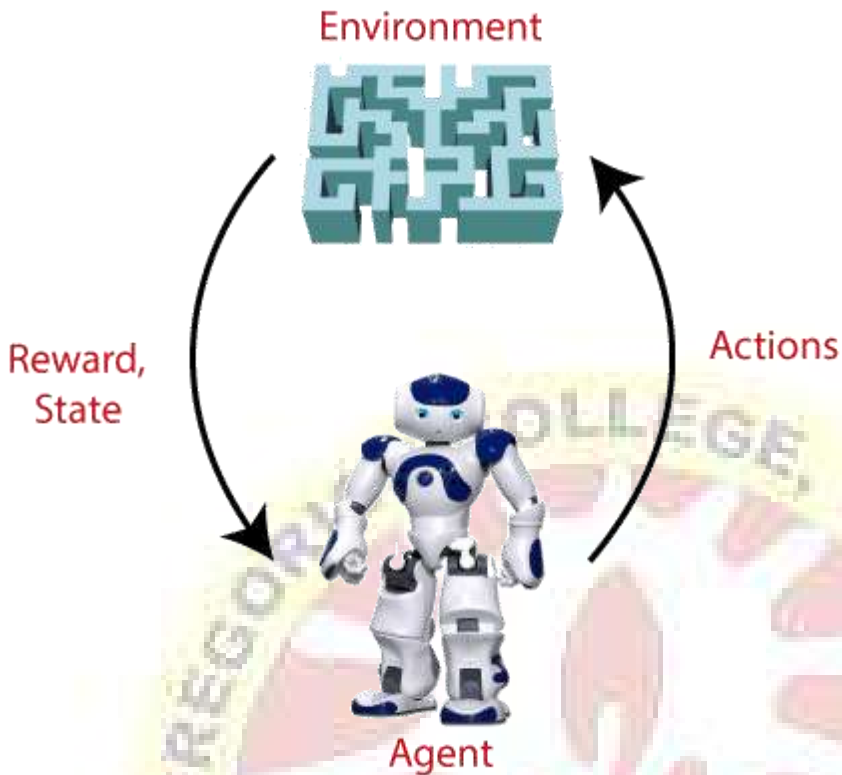
---

### **What is Reinforcement Learning?**

- Reinforcement Learning is a feedback-based Machine learning technique in which an agent learns to behave in an environment by performing the actions and seeing the results of actions. For each good action, the agent gets positive feedback, and for each bad action, the agent gets negative feedback or penalty.
- In Reinforcement Learning, the agent learns automatically using feedbacks without any labeled data, unlike [supervised learning.](#)
- Since there is no labeled data, so the agent is bound to learn by its experience only.
- RL solves a specific type of problem where decision making is sequential, and the goal is long-term, such as **game-playing, robotics**, etc.

- The agent interacts with the environment and explores it by itself. The primary goal of an agent in reinforcement learning is to improve the performance by getting the maximum positive rewards.
- The agent learns with the process of hit and trial, and based on the experience, it learns to perform the task in a better way. Hence, we can say that *"Reinforcement learning is a type of machine learning method where an intelligent agent (computer program) interacts with the environment and learns to act within that."* How a Robotic dog learns the movement of his arms is an example of Reinforcement learning.
- It is a core part of [Artificial intelligence](#), and all [AI agent](#) works on the concept of reinforcement learning. Here we do not need to pre-program the agent, as it learns from its own experience without any human intervention.
- **Example:** Suppose there is an AI agent present within a maze environment, and his goal is to find the diamond. The agent interacts with the environment by performing some actions, and based on those actions, the state of the agent gets changed, and it also receives a reward or penalty as feedback.
- The agent continues doing these three things (**take action, change state/remain in the same state, and get feedback**), and by doing these actions, he learns and explores the environment.
- The agent learns that what actions lead to positive feedback or rewards and what actions lead to negative feedback penalty. As a positive reward, the agent gets a positive point, and as a penalty, it gets a negative point.





### Terms used in Reinforcement Learning

- **Agent():** An entity that can perceive/explore the environment and act upon it.
- **Environment():** A situation in which an agent is present or surrounded by. In RL, we assume the stochastic environment, which means it is random in nature.
- **Action():** Actions are the moves taken by an agent within the environment.
- **State():** State is a situation returned by the environment after each action taken by the agent.
- **Reward():** A feedback returned to the agent from the environment to evaluate the action of the agent.
- **Policy():** Policy is a strategy applied by the agent for the next action based on the current state.
- **Value():** It is expected long-term returned with the discount factor and opposite to the short-term reward.



- **Q-value():** It is mostly similar to the value, but it takes one additional parameter as a current action (a).

---

### Key Features of Reinforcement Learning

- In RL, the agent is not instructed about the environment and what actions need to be taken.
- It is based on the hit and trial process.
- The agent takes the next action and changes states according to the feedback of the previous action.
- The agent may get a delayed reward.
- The environment is stochastic, and the agent needs to explore it to reach to get the maximum positive rewards.

---

### Approaches to implement Reinforcement Learning

There are mainly three ways to implement reinforcement-learning in ML, which are:

#### 1. Value-based:

The value-based approach is about to find the optimal value function, which is the maximum value at a state under any policy. Therefore, the agent expects the long-term return at any state(s) under policy  $\pi$ .

#### 2. Policy-based:

Policy-based approach is to find the optimal policy for the maximum future rewards without using the value function. In this approach, the agent tries to apply such a policy that the action performed in each step helps to maximize the future reward.

The policy-based approach has mainly two types of policy:

- **Deterministic:** The same action is produced by the policy ( $\pi$ ) at any state.
- **Stochastic:** In this policy, probability determines the produced action.

3. **Model-based:** In the model-based approach, a virtual model is created for the environment, and the agent explores that environment to learn it. There is no particular solution or algorithm for this approach because the model representation is different for each environment.

## Elements of Reinforcement Learning

There are four main elements of Reinforcement Learning, which are given below:

1. Policy
2. Reward Signal
3. Value Function
4. Model of the environment

**1) Policy:** A policy can be defined as a way how an agent behaves at a given time. It maps the perceived states of the environment to the actions taken on those states. A policy is the core element of the RL as it alone can define the behavior of the agent. In some cases, it may be a simple function or a lookup table, whereas, for other cases, it may involve general computation as a search process. It could be deterministic or a stochastic policy:

**For deterministic policy:**  $a = \pi(s)$

**For stochastic policy:**  $\pi(a | s) = P[At = a | St = s]$

**2) Reward Signal:** The goal of reinforcement learning is defined by the reward signal. At each state, the environment sends an immediate signal to the learning agent, and this signal is known as a **reward signal**. These rewards are given according to the good and bad actions taken by the agent. The agent's main objective is to maximize the total number of rewards for good actions. The reward signal can change the policy, such as if an action selected by the agent leads to low reward, then the policy may change to select other actions in the future.

**3) Value Function:** The value function gives information about how good the situation and action are and how much reward an agent can expect. A reward indicates the **immediate signal**

**for each good and bad action**, whereas a value function specifies **the good state and action for the future**. The value function depends on the reward as, without reward, there could be no value. The goal of estimating values is to achieve more rewards.

**4) Model:** The last element of reinforcement learning is the model, which mimics the behavior of the environment. With the help of the model, one can make inferences about how the environment will behave. Such as, if a state and an action are given, then a model can predict the next state and reward.

The model is used for planning, which means it provides a way to take a course of action by considering all future situations before actually experiencing those situations. The approaches for solving the RL problems **with the help of the model** are termed as the **model-based approach**. Comparatively, an approach **without using a model** is called a **model-free approach**.

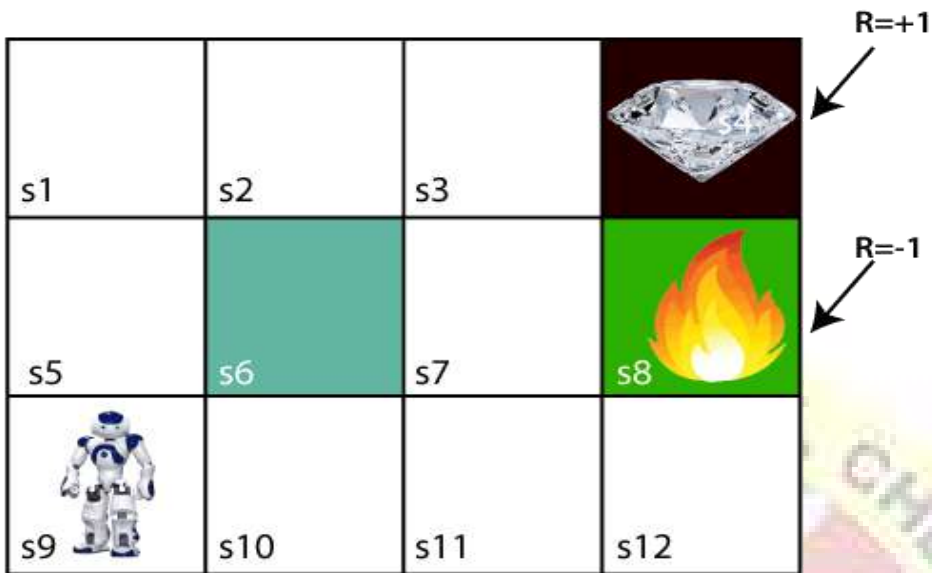
### How does Reinforcement Learning Work?

To understand the working process of the RL, we need to consider two main things:

- **Environment:** It can be anything such as a room, maze, football ground, etc.
- **Agent:** An intelligent agent such as AI robot.

Let's take an example of a maze environment that the agent needs to explore. Consider the below image:









Lorem ipsum

In the above image, the agent is at the very first block of the maze. The maze is consisting of an  $S_6$  block, which is a **wall**,  $S_8$  a **fire pit**, and  $S_4$  a **diamond block**.

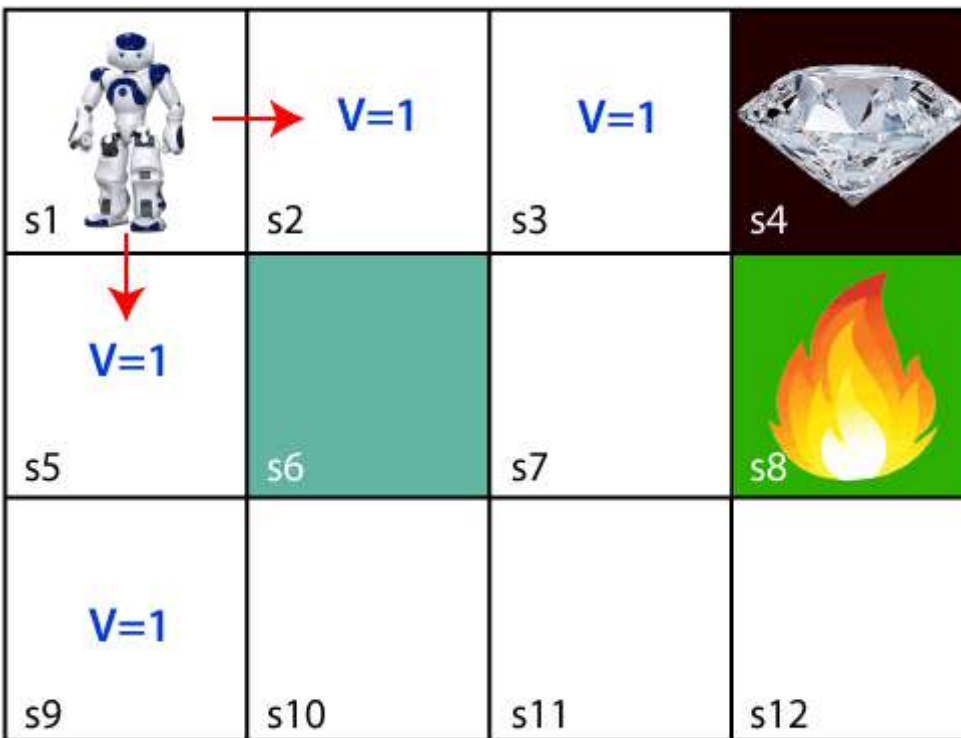
The agent cannot cross the  $S_6$  block, as it is a solid wall. If the agent reaches the  $S_4$  block, then get the **+1 reward**; if it reaches the fire pit, then gets **-1 reward point**. It can take four actions: **move up, move down, move left, and move right**.

The agent can take any path to reach to the final point, but he needs to make it in possible fewer steps. Suppose the agent considers the path **S9-S5-S1-S2-S3**, so he will get the +1-reward point.

The agent will try to remember the preceding steps that it has taken to reach the final step. To memorize the steps, it assigns 1 value to each previous step. Consider the below step:

V=1 s1	V=1 s2	V=1 s3	 s4
V=1 s5	 s6	s7	 s8
 V=1 s9	s10	s11	s12

Now, the agent has successfully stored the previous steps assigning the 1 value to each previous block. But what will the agent do if he starts moving from the block, which has 1 value block on both sides? Consider the below diagram:



It will be a difficult condition for the agent whether he should go up or down as each block has the same value. So, the above approach is not suitable for the agent to reach the destination. Hence to solve the problem, we will use the **Bellman equation**, which is the main concept behind reinforcement learning.

### The Bellman Equation

The Bellman equation was introduced by the Mathematician **Richard Ernest Bellman in the year 1953**, and hence it is called as a Bellman equation. It is associated with dynamic programming and used to calculate the values of a decision problem at a certain point by including the values of previous states.

It is a way of calculating the value functions in dynamic programming or environment that leads to modern reinforcement learning.

The key-elements used in Bellman equations are:

- Action performed by the agent is referred to as "a"
- State occurred by performing the action is "s."
- The reward/feedback obtained for each good and bad action is "R."
- A discount factor is Gamma " $\gamma$ ."

The Bellman equation can be written as:

1.  $V(s) = \max [R(s,a) + \gamma V(s')]$

Where,

**$V(s)$  = value calculated at a particular point.**

**$R(s,a)$  = Reward at a particular state  $s$  by performing an action.**

**$\gamma$  = Discount factor**

**$V(s')$  = The value at the previous state.**

In the above equation, we are taking the max of the complete values because the agent tries to find the optimal solution always.

So now, using the Bellman equation, we will find value at each state of the given environment. We will start from the block, which is next to the target block.

**For 1st block:**

$V(s_3) = \max [R(s,a) + \gamma V(s')]$ , here  $V(s') = 0$  because there is no further state to move.

$V(s_3) = \max [R(s,a)] \Rightarrow V(s_3) = \max [1] \Rightarrow V(s_3) = 1.$

**For 2nd block:**

$V(s_2) = \max [R(s,a) + \gamma V(s')]$ , here  $\gamma = 0.9$  (lets),  $V(s') = 1$ , and  $R(s, a) = 0$ , because there is no reward at this state.

$$V(s_2) = \max[0.9(1)] \Rightarrow V(s) = \max[0.9] \Rightarrow V(s_2) = \mathbf{0.9}$$

**For 3rd block:**

$V(s_1) = \max [R(s,a) + \gamma V(s')]$ , here  $\gamma = 0.9$  (lets),  $V(s') = 0.9$ , and  $R(s, a) = 0$ , because there is no reward at this state also.

$$V(s_1) = \max[0.9(0.9)] \Rightarrow V(s_3) = \max[0.81] \Rightarrow V(s_1) = \mathbf{0.81}$$

**For 4th block:**

$V(s_5) = \max [R(s,a) + \gamma V(s')]$ , here  $\gamma = 0.9$  (lets),  $V(s') = 0.81$ , and  $R(s, a) = 0$ , because there is no reward at this state also.

$$V(s_5) = \max[0.9(0.81)] \Rightarrow V(s_5) = \max[0.81] \Rightarrow V(s_5) = \mathbf{0.73}$$

**For 5th block:**





$V(s_9) = \max [R(s,a) + \gamma V(s')]$ , here  $\gamma = 0.9$  (lets),  $V(s') = 0.73$ , and  $R(s, a) = 0$ , because there is no reward at this state also.

$$V(s_9) = \max[0.9(0.73)] \Rightarrow V(s_4) = \max[0.81] \Rightarrow V(s_4) = \mathbf{0.66}$$





**Consider the below image:**





V=0.81 s1	V=0.9 s2	V=1 s3	 s4
V=0.73 s5		s7	 s8
 V=0.66 s9	s10	s11	s12



Now, we will move further to the 6<sup>th</sup> block, and here agent may change the route because it always tries to find the optimal path. So now, let's consider from the block next to the fire pit.

V=0.81 s1	V=0.9 s2	V=1 s3	 s4
V=0.73 s5		 s7	 s8
V=0.66 s9	s10	s11	s12

Red arrows indicate movement directions: up from s3, down from s7, and right from s7.

Now, the agent has three options to move; if he moves to the blue box, then he will feel a bump if he moves to the fire pit, then he will get the -1 reward. But here we are taking only positive rewards, so for this, he will move to upwards only. The complete block values will be calculated using this formula. Consider the below image:

LET YOUR LIGHT SHINE

V=0.81 s1	V=0.9 s2	V=1 s3	 s4
V=0.73 s5	s6	V=0.9 s7	 s8
V=0.66 s9	V=0.73 s10	V=0.81 s11	V=0.73 s12

### Types of Reinforcement learning

There are mainly two types of reinforcement learning, which are:

- **Positive Reinforcement**
- **Negative Reinforcement**

#### **Positive Reinforcement:**

The positive reinforcement learning means adding something to increase the tendency that expected behavior would occur again. It impacts positively on the behavior of the agent and increases the strength of the behavior.

This type of reinforcement can sustain the changes for a long time, but too much positive reinforcement may lead to an overload of states that can reduce the consequences.

### Negative Reinforcement:

The negative reinforcement learning is opposite to the positive reinforcement as it increases the tendency that the specific behavior will occur again by avoiding the negative condition.

It can be more effective than the positive reinforcement depending on situation and behavior, but it provides reinforcement only to meet minimum behavior.

### How to represent the agent state?

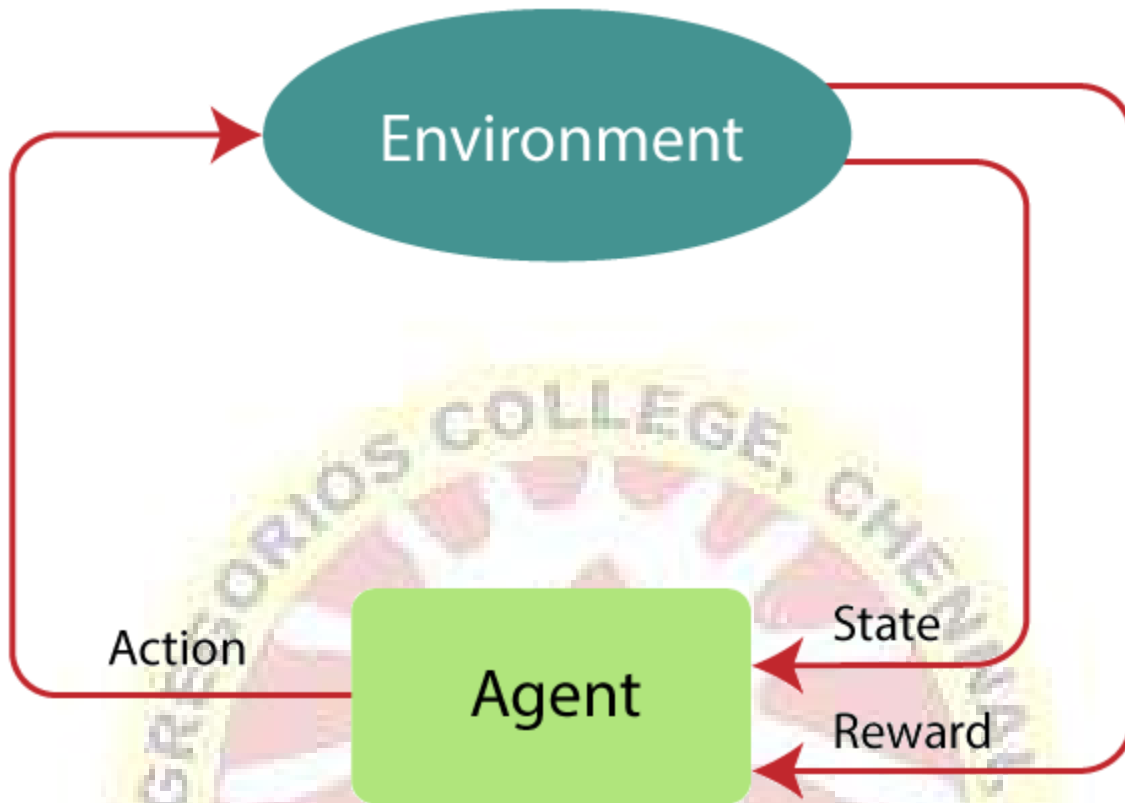
We can represent the agent state using the **Markov State** that contains all the required information from the history. The State  $S_t$  is Markov state if it follows the given condition:

$$P[S_{t+1} | S_t] = P[S_{t+1} | S_1, \dots, S_t]$$

The Markov state follows the **Markov property**, which says that the future is independent of the past and can only be defined with the present. The RL works on fully observable environments, where the agent can observe the environment and act for the new state. The complete process is known as Markov Decision process, which is explained below:

### Markov Decision Process

Markov Decision Process or MDP, is used to **formalize the reinforcement learning problems**. If the environment is completely observable, then its dynamic can be modeled as a **Markov Process**. In MDP, the agent constantly interacts with the environment and performs actions; at each action, the environment responds and generates a new state.



MDP is used to describe the environment for the RL, and almost all the RL problem can be formalized using MDP.

MDP contains a tuple of four elements  $(S, A, P_a, R_a)$ :

- A set of finite States  $S$
- A set of finite Actions  $A$
- Rewards received after transitioning from state  $S$  to state  $S'$ , due to action  $a$ .
- Probability  $P_a$ .

MDP uses **Markov property**, and to better understand the MDP, we need to learn about it.

### Markov Property:

It says that *"If the agent is present in the current state  $S_1$ , performs an action  $a_1$  and move to the state  $s_2$ , then the state transition from  $s_1$  to  $s_2$  only depends on the current state and future action and states do not depend on past actions, rewards, or states."*

Or, in other words, as per Markov Property, the current state transition does not depend on any past action or state. Hence, MDP is an RL problem that satisfies the Markov property. Such as in a **Chess game, the players only focus on the current state and do not need to remember past actions or states.**

### Finite MDP:

A finite MDP is when there are finite states, finite rewards, and finite actions. In RL, we consider only the finite MDP.

### Markov Process:

Markov Process is a memoryless process with a sequence of random states  $S_1, S_2, \dots, S_t$  that uses the Markov Property. Markov process is also known as Markov chain, which is a tuple  $(S, P)$  on state  $S$  and transition function  $P$ . These two components ( $S$  and  $P$ ) can define the dynamics of the system.

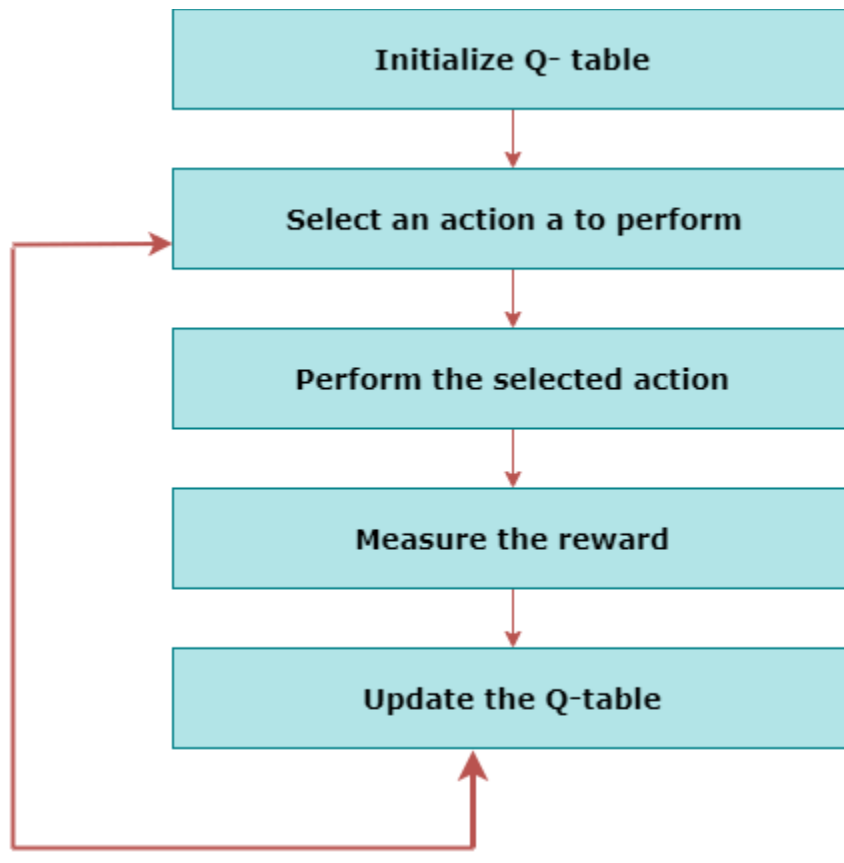
---

## Reinforcement Learning Algorithms

Reinforcement learning algorithms are mainly used in AI applications and gaming applications. The main used algorithms are:

- **Q-Learning:**
  - Q-learning is an **Off policy RL algorithm**, which is used for the temporal difference Learning. The temporal difference learning methods are the way of comparing temporally successive predictions.

- It learns the value function  $Q(S, a)$ , which means how good to take action "a" at a particular state "s."
- The below flowchart explains the working of Q- learning:



- **State Action Reward State action (SARSA):**

- SARSA stands for **State Action Reward State action**, which is an **on-policy** temporal difference learning method. The on-policy control method selects the action for each state while learning using a specific policy.
- The goal of SARSA is to calculate the  **$Q \pi (s, a)$  for the selected current policy  $\pi$  and all pairs of (s-a).**
- The main difference between Q-learning and SARSA algorithms is that **unlike Q-learning, the maximum reward for the next state is not required for updating the Q-value in the table.**
- In SARSA, new action and reward are selected using the same policy, which has determined the original action.

- The SARSA is named because it uses the quintuple  $Q(s, a, r, s', a')$ . Where,
  - s:** original state
  - a:** Original action
  - r:** reward observed while following the states
  - s' and a':** New state, action pair.
- **Deep Q Neural Network (DQN):**
  - As the name suggests, DQN is a **Q-learning using Neural networks**.
  - For a big state space environment, it will be a challenging and complex task to define and update a Q-table.
  - To solve such an issue, we can use a DQN algorithm. Where, instead of defining a Q-table, neural network approximates the Q-values for each action and state.

Now, we will expand the Q-learning.

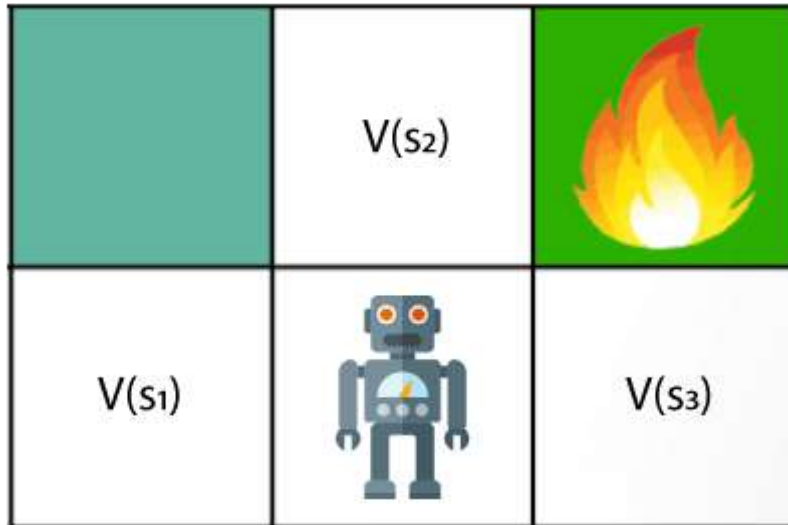
### Q-Learning Explanation:

- Q-learning is a popular model-free reinforcement learning algorithm based on the Bellman equation.
- **The main objective of Q-learning is to learn the policy which can inform the agent that what actions should be taken for maximizing the reward under what circumstances.**
- It is an **off-policy RL** that attempts to find the best action to take at a current state.
- The goal of the agent in Q-learning is to maximize the value of Q.
- The value of Q-learning can be derived from the Bellman equation. Consider the Bellman equation given below:

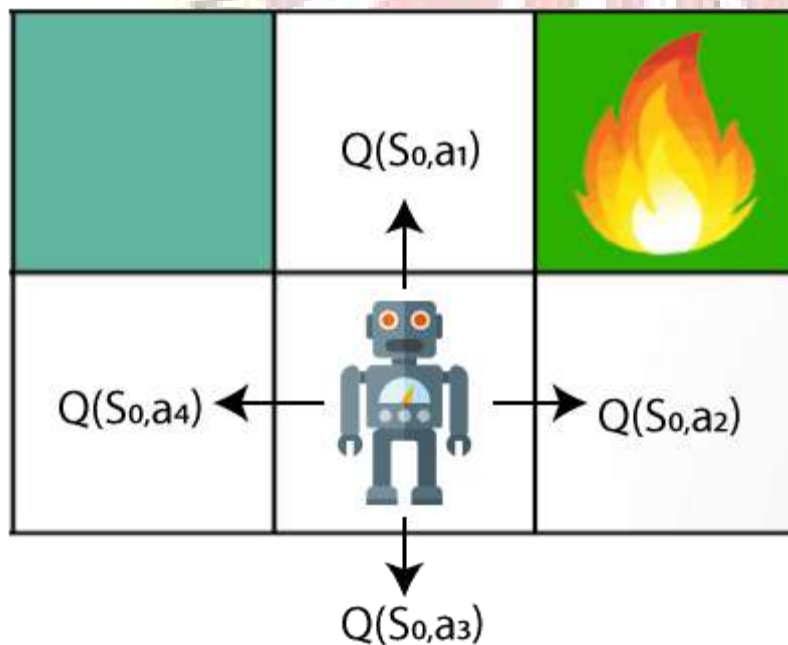
$$V(s) = \max [R(s,a) + \gamma \sum_{s'} P(s, a, s') V(s')]$$

In the equation, we have various components, including reward, discount factor ( $\gamma$ ), probability, and end states  $s'$ . But there is no any Q-value is given so first consider the below image:





In the above image, we can see there is an agent who has three values options,  $V(s_1)$ ,  $V(s_2)$ ,  $V(s_3)$ . As this is MDP, so agent only cares for the current state and the future state. The agent can go to any direction (Up, Left, or Right), so he needs to decide where to go for the optimal path. Here agent will take a move as per probability bases and changes the state. But if we want some exact moves, so for this, we need to make some changes in terms of Q-value. Consider the below image:



Q- represents the quality of the actions at each state. So instead of using a value at each state, we will use a pair of state and action, i.e.,  $Q(s, a)$ . Q-value specifies that which action is more lubricative than others, and according to the best Q-value, the agent takes his next move. The Bellman equation can be used for deriving the Q-value.

To perform any action, the agent will get a reward  $R(s, a)$ , and also he will end up on a certain state, so the Q -value equation will be:

$$Q(S, a) = R(s, a) + \gamma \sum_{s'} P(s, a, s') V(s')$$

Hence, we can say that,  $V(s) = \max [Q(s, a)]$

$$Q(S, a) = R(s, a) + \gamma \sum_{s'} (P(s, a, s') \max_{a'} Q(s', a'))$$

**The above formula is used to estimate the Q-values in Q-Learning.**

**What is 'Q' in Q-learning?**

The Q stands for **quality** in **Q-learning**, which means it specifies the quality of an action taken by the agent.

**Q-table:**

A Q-table or matrix is created while performing the Q-learning. The table follows the state and action pair, i.e.,  $[s, a]$ , and initializes the values to zero. After each action, the table is updated, and the q-values are stored within the table.

The RL agent uses this Q-table as a reference table to select the best action based on the q-values.

---

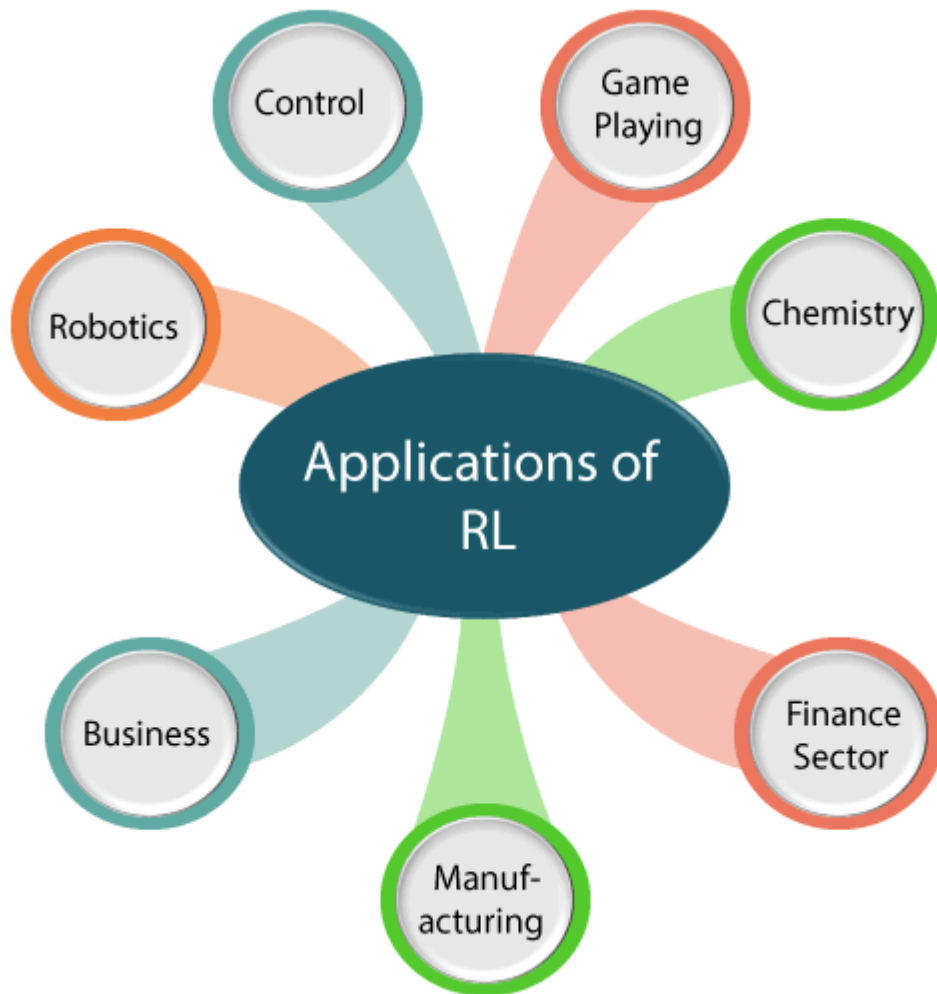
## Difference between Reinforcement Learning and Supervised Learning

The Reinforcement Learning and Supervised Learning both are the part of machine learning, but both types of learnings are far opposite to each other. The RL agents interact with the environment, explore it, take action, and get rewarded. Whereas supervised learning algorithms learn from the labeled dataset and, on the basis of the training, predict the output.

The difference table between RL and Supervised learning is given below:

Reinforcement Learning	Supervised Learning
RL works by interacting with the environment.	Supervised learning works on the existing dataset.
The RL algorithm works like the human brain works when making some decisions.	Supervised Learning works as when a human learns things in the supervision of a guide.
There is no labeled dataset is present	The labeled dataset is present.
No previous training is provided to the learning agent.	Training is provided to the algorithm so that it can predict the output.
RL helps to take decisions sequentially.	In Supervised learning, decisions are made when input is given.

## Reinforcement Learning Applications



### 1. Robotics:

- a. RL is used in **Robot navigation, Robo-soccer, walking, juggling**, etc.

#### **Control:**

- . RL can be used for **adaptive control** such as Factory processes, admission control in telecommunication, and Helicopter pilot is an example of reinforcement learning.

#### **Game Playing:**

- . RL can be used in **Game playing** such as tic-tac-toe, chess, etc.

#### **Chemistry:**

- . RL can be used for optimizing the chemical reactions.

**Business:**

- . RL is now used for business strategy planning.

**Manufacturing:**

- . In various automobile manufacturing companies, the robots use deep reinforcement learning to pick goods and put them in some containers.

**Finance Sector:**

- . The RL is currently used in the finance sector for evaluating trading strategies.

**UNIT V**

Robotics is a domain in artificial intelligence that deals with the study of creating intelligent and efficient robots.

**What are Robots?**

Robots are the artificial agents acting in real world environment.

**Objective**

Robots are aimed at manipulating the objects by perceiving, picking, moving, modifying the physical properties of object, destroying it, or to have an effect thereby freeing manpower from doing repetitive functions without getting bored, distracted, or exhausted.

**What is Robotics?**

Robotics is a branch of AI, which is composed of Electrical Engineering, Mechanical Engineering, and Computer Science for designing, construction, and application of robots.

**Aspects of Robotics**

- The robots have **mechanical construction**, form, or shape designed to accomplish a particular task.
- They have **electrical components** which power and control the machinery.

- They contain some level of **computer program** that determines what, when and how a robot does something.

### Difference in Robot System and Other AI Program

Here is the difference between the two –

AI Programs	Robots
They usually operate in computer-stimulated worlds.	They operate in real physical world
The input to an AI program is in symbols and rules.	Inputs to robots is analog signal in the form of speech waveform or images
They need general purpose computers to operate on.	They need special hardware with sensors and effectors.

### Robot Locomotion

Locomotion is the mechanism that makes a robot capable of moving in its environment. There are various types of locomotions –

- Legged
- Wheeled
- Combination of Legged and Wheeled Locomotion
- Tracked slip/skid

## Legged Locomotion

- This type of locomotion consumes more power while demonstrating walk, jump, trot, hop, climb up or down, etc.
- It requires more number of motors to accomplish a movement. It is suited for rough as well as smooth terrain where irregular or too smooth surface makes it consume more power for a wheeled locomotion. It is little difficult to implement because of stability issues.
- It comes with the variety of one, two, four, and six legs. If a robot has multiple legs then leg coordination is necessary for locomotion.

The total number of possible **gaits** (a periodic sequence of lift and release events for each of the total legs) a robot can travel depends upon the number of its legs.

If a robot has  $k$  legs, then the number of possible events  $N = (2k-1)!$ .

In case of a two-legged robot ( $k=2$ ), the number of possible events is  $N = (2k-1)! = (2*2-1)! = 3! = 6$ .

Hence there are six possible different events –

- Lifting the Left leg
- Releasing the Left leg
- Lifting the Right leg
- Releasing the Right leg
- Lifting both the legs together
- Releasing both the legs together

In case of  $k=6$  legs, there are 39916800 possible events. Hence the complexity of robots is directly proportional to the number of legs.



### Wheeled Locomotion

It requires fewer number of motors to accomplish a movement. It is little easy to implement as there are less stability issues in case of more number of wheels. It is power efficient as compared to legged locomotion.

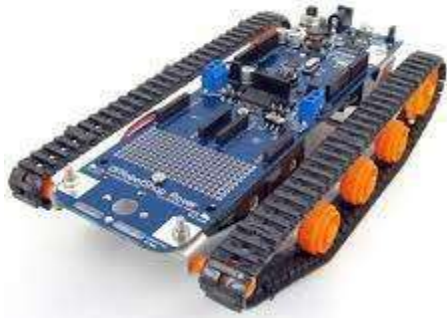
- **Standard wheel** – Rotates around the wheel axle and around the contact
- **Castor wheel** – Rotates around the wheel axle and the offset steering joint.
- **Swedish 45° and Swedish 90° wheels** – Omni-wheel, rotates around the contact point, around the wheel axle, and around the rollers.
- **Ball or spherical wheel** – Omnidirectional wheel, technically difficult to implement.





### Slip/Skid Locomotion

In this type, the vehicles use tracks as in a tank. The robot is steered by moving the tracks with different speeds in the same or opposite direction. It offers stability because of large contact area of track and ground.



### Components of a Robot

Robots are constructed with the following –

- **Power Supply** – The robots are powered by batteries, solar power, hydraulic, or pneumatic power sources.
- **Actuators** – They convert energy into movement.
- **Electric motors (AC/DC)** – They are required for rotational movement.
- **Pneumatic Air Muscles** – They contract almost 40% when air is sucked in them.
- **Muscle Wires** – They contract by 5% when electric current is passed through them.
- **Piezo Motors and Ultrasonic Motors** – Best for industrial robots.
- **Sensors** – They provide knowledge of real time information on the task environment. Robots are equipped with vision sensors to be to compute the depth in the environment. A tactile sensor imitates the mechanical properties of touch receptors of human fingertips.

## Computer Vision

This is a technology of AI with which the robots can see. The computer vision plays vital role in the domains of safety, security, health, access, and entertainment.

Computer vision automatically extracts, analyzes, and comprehends useful information from a single image or an array of images. This process involves development of algorithms to accomplish automatic visual comprehension.

## Hardware of Computer Vision System

This involves –

- Power supply
- Image acquisition device such as camera
- A processor
- A software
- A display device for monitoring the system
- Accessories such as camera stands, cables, and connectors

## Tasks of Computer Vision

- **OCR** – In the domain of computers, Optical Character Reader, a software to convert scanned documents into editable text, which accompanies a scanner.
- **Face Detection** – Many state-of-the-art cameras come with this feature, which enables to read the face and take the picture of that perfect expression. It is used to let a user access the software on correct match.
- **Object Recognition** – They are installed in supermarkets, cameras, high-end cars such as BMW, GM, and Volvo.
- **Estimating Position** – It is estimating position of an object with respect to camera as in position of tumor in human's body.

## Application Domains of Computer Vision

- Agriculture

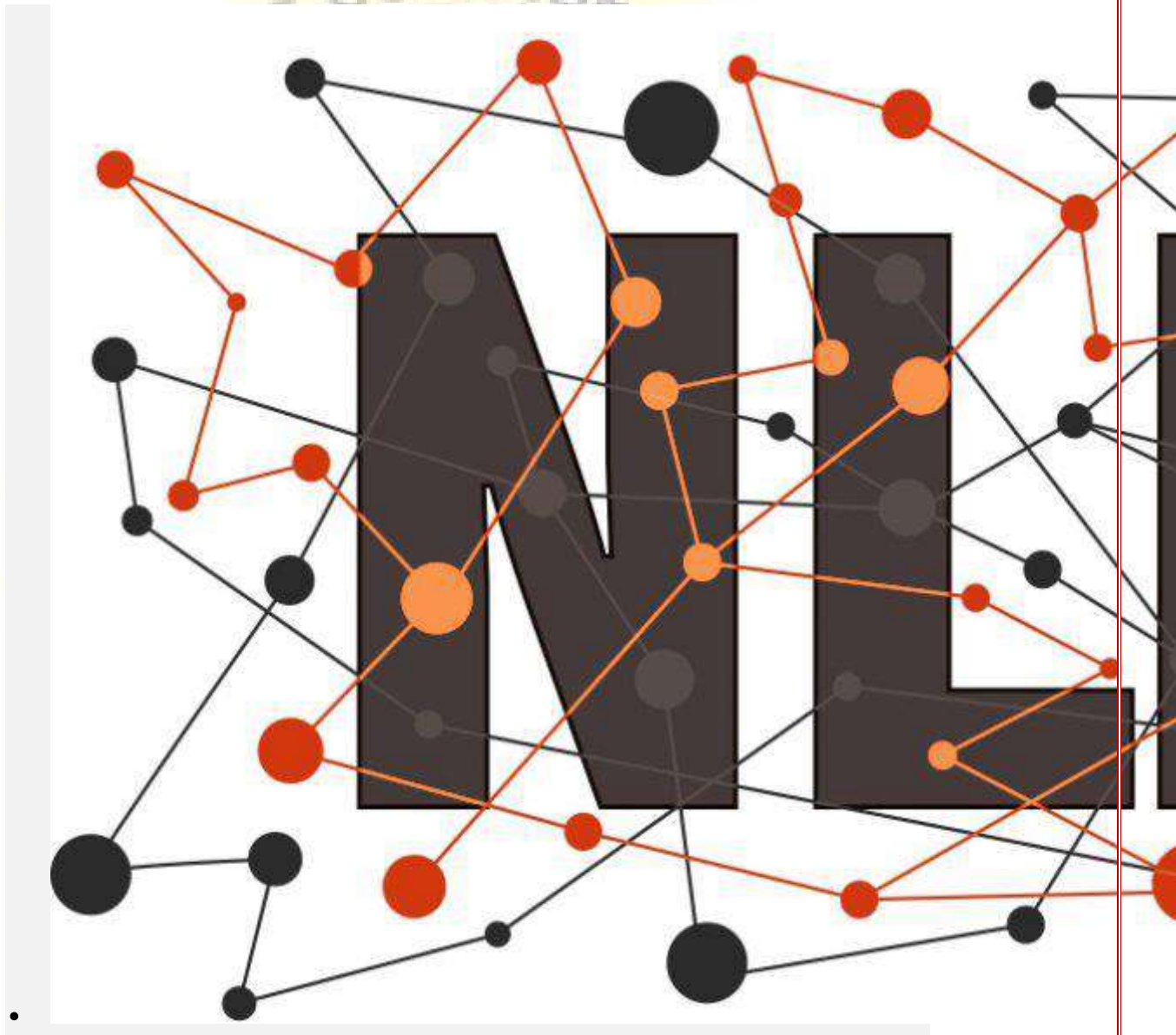
- Autonomous vehicles
- Biometrics
- Character recognition
- Forensics, security, and surveillance
- Industrial quality inspection
- Face recognition
- Gesture analysis
- Geoscience
- Medical imagery
- Pollution monitoring
- Process control
- Remote sensing
- Robotics
- Transport

### Applications of Robotics

The robotics has been instrumental in the various domains such as –

- **Industries** – Robots are used for handling material, cutting, welding, color coating, drilling, polishing, etc.
- **Military** – Autonomous robots can reach inaccessible and hazardous zones during war. A robot named *Daksh*, developed by Defense Research and Development Organization (DRDO), is in function to destroy life-threatening objects safely.
- **Medicine** – The robots are capable of carrying out hundreds of clinical tests simultaneously, rehabilitating permanently disabled people, and performing complex surgeries such as brain tumors.
- **Exploration** – The robot rock climbers used for space exploration, underwater drones used for ocean exploration are to name a few.
- **Entertainment** – Disney’s engineers have created hundreds of robots for movie making.

Language Processing in “A Neural Probabilistic Language Model:”



- Credit: smartdatacollective.com

- This is the PLN (plan): discuss NLP (Natural Language Processing) seen through the lens of probability, in a model put forth by Bengio et al. in 2003 called NPL (Neural Probabilistic Language). The year the paper was published is important to consider at the get-go because it was a **fulcrum** moment in the history of how we analyze human language using computers. Noam Chomsky's Linguistics might be seen as an effort to use the human mind like a machine and systematically break down language into smaller and smaller components. He started with sentences and went to words, then to morphemes and finally phonemes. Computerization takes this powerful concept and makes it into something even more vital to humankind: it starts with being relevant to individuals and goes to teams of people, then to corporations and finally governments. Dr. Chomsky truly changed the way we approach communication, and that influence can still be felt. Linguistics was powerful when it was first introduced, and it is powerful today. N-gram analysis, or any kind of computational linguistics for that matter, are derived from the work of this great man, this forerunner. This blog will summarize the work of the Bengio group, thought leaders who took up the torch of knowledge to advance our understanding of natural language and how computers interact with it.
- Artificial Intelligence has changed considerably since 2003, but the model presented in this paper captures the essence of why it was able to take off. Machine learning and deep learning have both become part of the AI canon since this paper was published, and as computing power continues to grow they are becoming ever more important. Data Science is a confluence of fields, and today we'll examine one which is a cornerstone of the discipline: probability. The probabilistic distribution model put forth in this paper, in essence, is a major reason we have improved our capabilities to process our natural language to such wuthering heights.
- English, considered to have the most words of any alphabetic language, is a probability nightmare. The possibilities for sequencing word combinations in even the most basic of sentences is inconceivable. We are facing something known

as **the curse of dimensionality**. To make this more concrete, the authors offer the following:

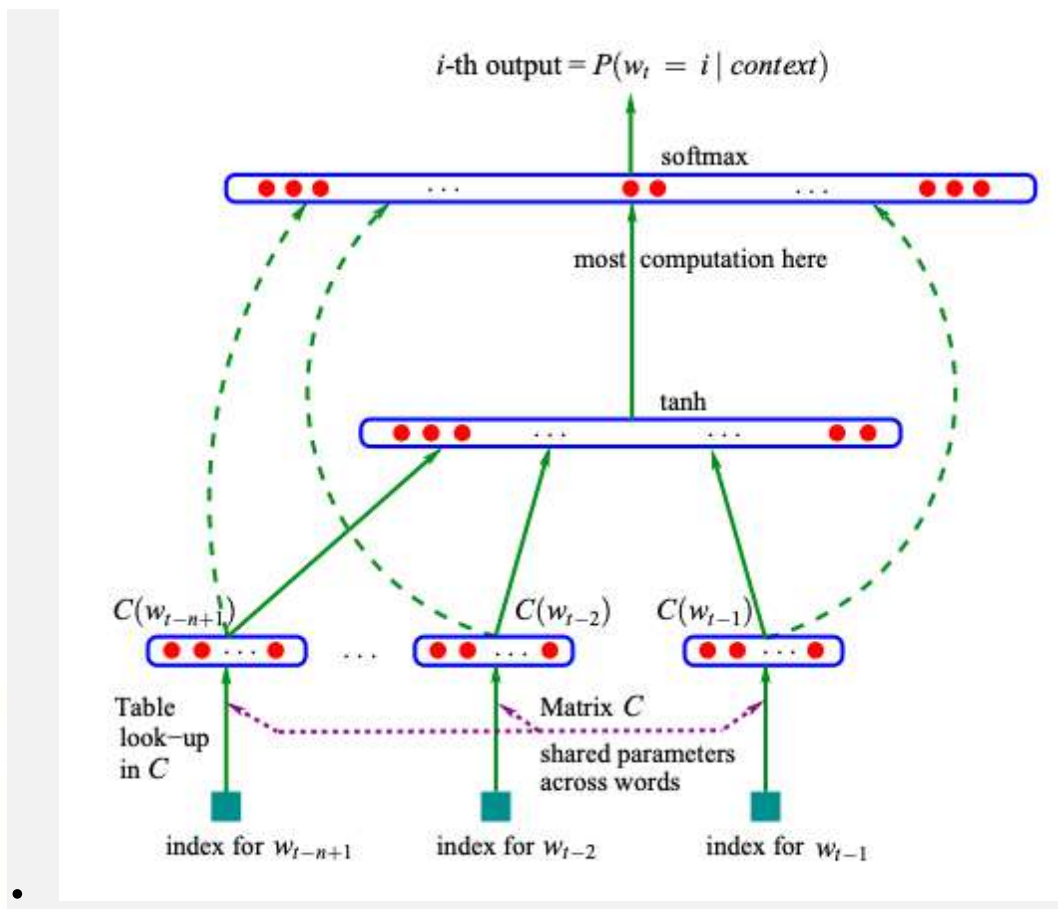
- *...if one wants to model the joint distribution of 10 consecutive words in a natural language with a vocabulary  $V$  of size 100,000, there are potentially  $100,000^{10} - 1 = 10^{50} - 1$  free parameters.*
- In data-driven Natural Language Processing tasks, there are practically unlimited discrete variables, because the population size of the English vocabulary is exponentially north of 100K. When trying to compare data that has been split into training and test sets, how can you ever expect to put forth a readily generalizable language model? The two divisions in your data are all but guaranteed to be vastly different, quite ungeneralizable. You're cursed by the amount of possibilities in the model, the amount of dimensions. What can be done?
- The Bengio group innovates not by using neural networks but by using them on a massive scale. Linguistics and its founding father Noam have a tendency to learn how one word interacts with all the others in a sentence. Bengio et al. focus on learning a statistical model of the distribution of word sequences. This research paper improves NLP firstly by considering not how a given word is similar to other words in the same sentence, but to new words that could fill the role of that given word. Secondly, they take into account n-gram approaches beyond unigram ( $n = 1$ ), bigram ( $n = 2$ ) or even trigram (the  $n$  typically used by researchers) up to an  $n$  of 5.
- The math for n-gram models is given as:



$$\hat{P}(w_t | w_1^{t-1}) \approx \hat{P}(w_t | w_{t-n+1}^{t-1}).$$

- 
- This formula is used to construct conditional probability tables for the next word to be predicted. When modeling NLP, the odds in the fight against dimensionality can be improved by taking advantage of word order, and by recognizing that temporally closer words in the word sequence are statistically more dependent. What does this ultimately mean in the context of what has been discussed? What problem is this solving? The language model proposed makes dimensionality less of a curse and more of an inconvenience. That is to say, computational and memory complexity scale up in a linear fashion, not exponentially. It improves upon past efforts by learning a feature vector for each word to represent similarity and also learning a probability function for how words connect via a neural network. Let's take a closer look at said neural network.

-

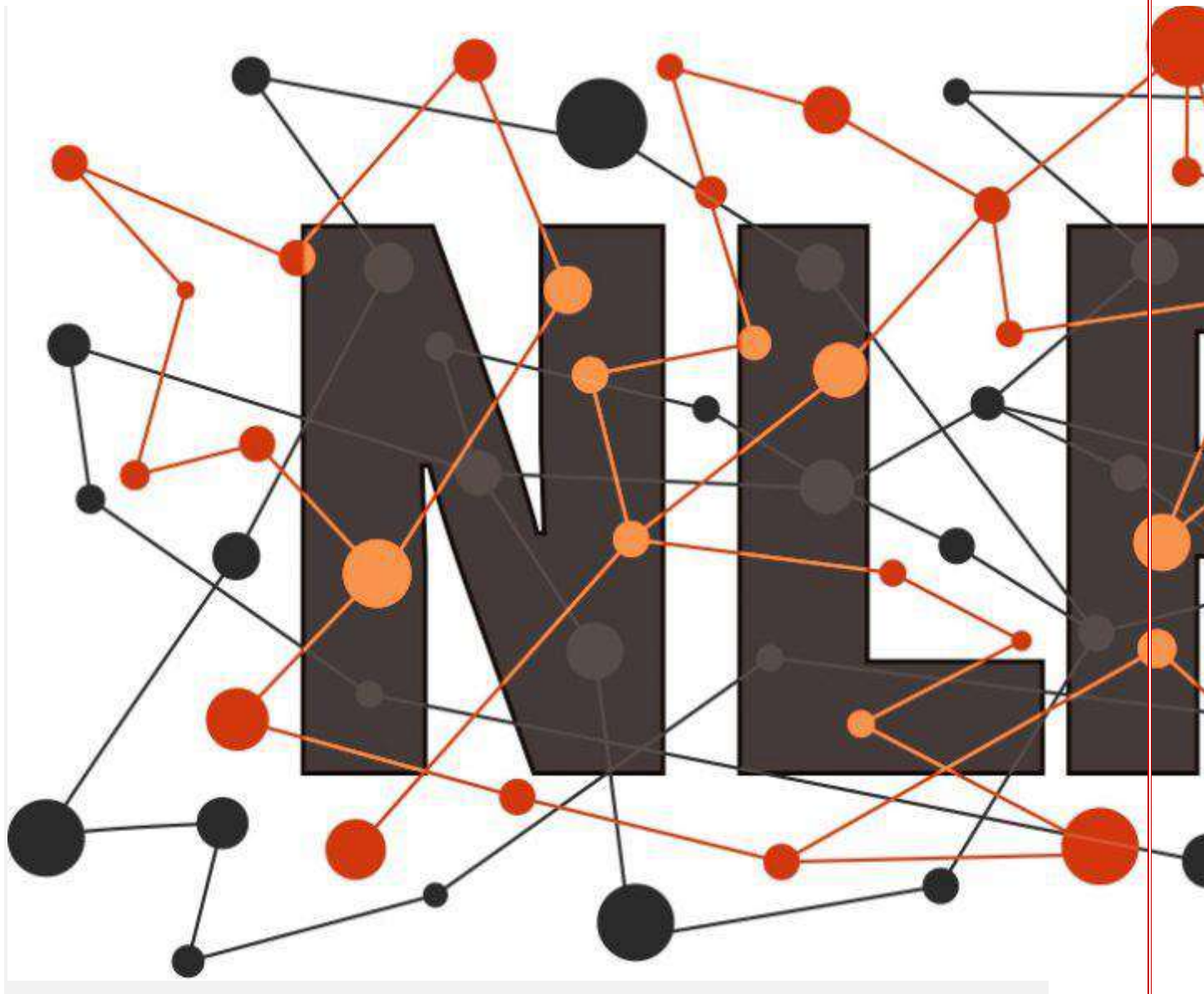


- We're presented here with something known as a Multi-Layer Perceptron. **What are those layers?** Three input nodes make up the foundation at the bottom, fed by the index for the word in the context of the text under study. The layer in the middle labeled  $\tanh$  represents the hidden layer.  $\tanh$ , an activation function known as the hyperbolic tangent, is sigmoidal (s-shaped) and helps reduce the chance of the model getting "stuck" when assigning values to the language being processed. **How is this?** In the system this research team sets up, strongly negative values get assigned values very close to -1 and vice versa for positive ones. Only zero-valued inputs are mapped to near-zero outputs. The uppermost layer is the output — the softmax function. It is used to bring our range of values into the probabilistic realm (in the interval from 0 to 1, in which all vector components sum up to 1).



- Don't overlook the dotted green lines connecting the inputs directly to outputs, either. The optional inclusion of this feature is brought up in the results section of the paper. It provides an interesting trade-off: including the direct connections between input and output causes the the training time to be cut in half (10 epochs to converge instead of 20). Without them, the model produced better generalizations via a tighter bottleneck formed in the hidden layer.
- It's possible for a sentence to obtain a high probability (even if the model has never encountered it before) if the words contained therein are similar to those in a previously observed one. There's the rub: Noam Chomsky and subsequent linguists are subject to criticisms of having developed too brittle of a system. This method sets the stage for a new kind of learning, deep learning. When utilized in conjunction with vector semantics, this is powerful stuff indeed. Through this paper, the Bengio team opened the door to the future and helped usher in a new era. An era of AI.

### **Language Processing in “A Neural Probabilistic Language Model:”**



Credit: smartdatacollective.com

This is the PLN (plan): discuss NLP (Natural Language Processing) seen through the lens of probability, in a model put forth by Bengio et al. in 2003 called NPL (Neural Probabilistic Language). The year the paper was published is important to consider at the get-go because it was a **fulcrum** moment in the history of how we analyze human language using computers. Noam Chomsky's Linguistics might be seen as an effort to use the human mind like a machine and systematically break down language into smaller and smaller components. He started with sentences and went to words, then to morphemes and finally phonemes. Computerization takes

this powerful concept and makes it into something even more vital to humankind: it starts with being relevant to individuals and goes to teams of people, then to corporations and finally governments. Dr. Chomsky truly changed the way we approach communication, and that influence can still be felt. Linguistics was powerful when it was first introduced, and it is powerful today. N-gram analysis, or any kind of computational linguistics for that matter, are derived from the work of this great man, this forerunner. This blog will summarize the work of the Bengio group, thought leaders who took up the torch of knowledge to advance our understanding of natural language and how computers interact with it.

Artificial Intelligence has changed considerably since 2003, but the model presented in this paper captures the essence of why it was able to take off. Machine learning and deep learning have both become part of the AI canon since this paper was published, and as computing power continues to grow they are becoming ever more important. Data Science is a confluence of fields, and today we'll examine one which is a cornerstone of the discipline: probability. The probabilistic distribution model put forth in this paper, in essence, is a major reason we have improved our capabilities to process our natural language to such wuthering heights.

English, considered to have the most words of any alphabetic language, is a probability nightmare. The possibilities for sequencing word combinations in even the most basic of sentences is inconceivable. We are facing something known as **the curse of dimensionality**. To make this more concrete, the authors offer the following:

*...if one wants to model the joint distribution of 10 consecutive words in a natural language with a vocabulary  $V$  of size 100,000, there are potentially  $100,000^{10} - 1 = 10^{50} - 1$  free parameters.*

In data-driven Natural Language Processing tasks, there are practically unlimited discrete variables, because the population size of the English vocabulary is exponentially north of 100K. When trying to compare data that has been split into training and test sets, how can you ever expect to put forth a readily generalizable language model? The two divisions in your data are all but guaranteed to be vastly different, quite ungeneralizable. You're cursed by the amount of possibilities in the model, the amount of dimensions. What can be done?

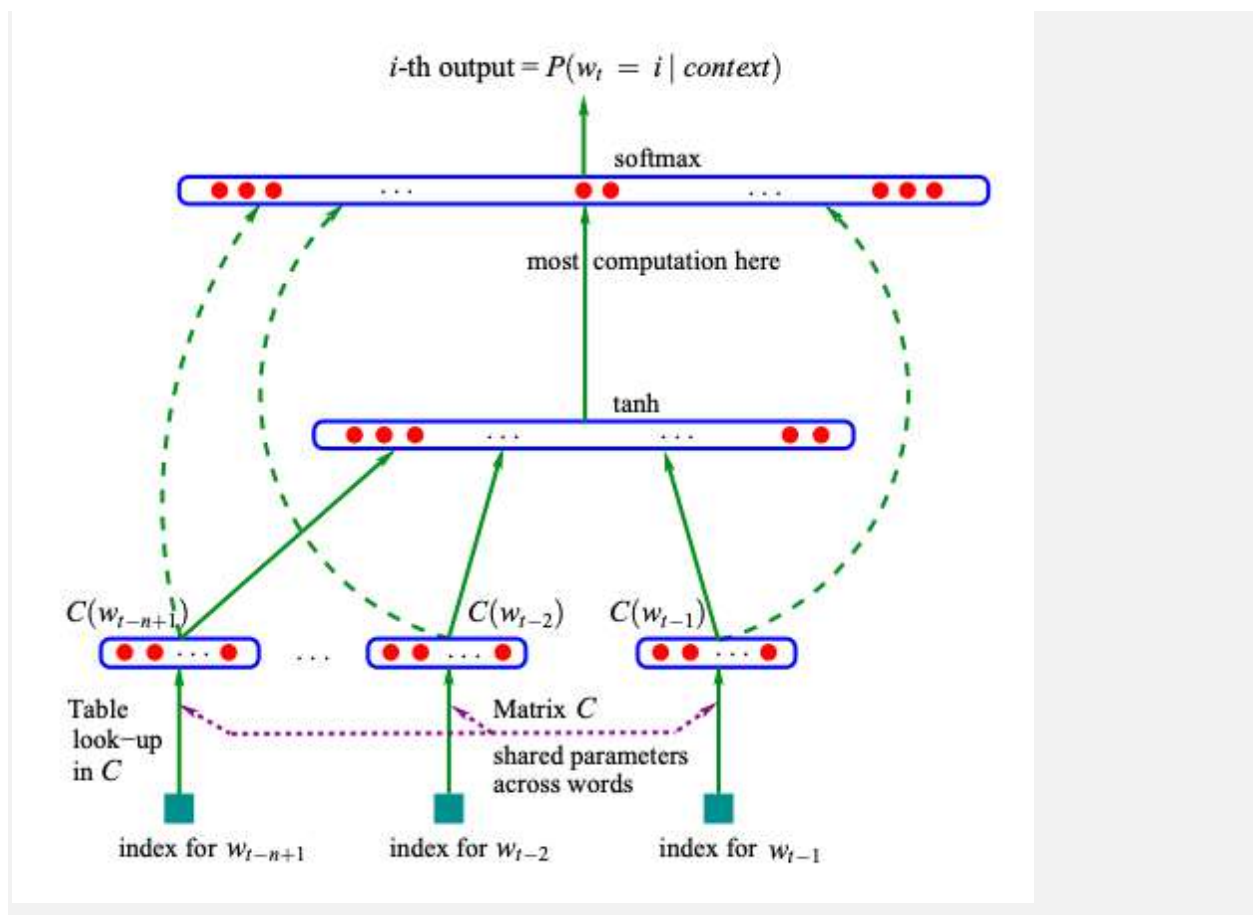
The Bengio group innovates not by using neural networks but by using them on a massive scale. Linguistics and its founding father Noam have a tendency to learn how one word interacts with all the others in a sentence. Bengio et al. focus on learning a statistical model of the distribution of word sequences. This research paper improves NLP firstly by considering not how a given word is similar to other words in the same sentence, but to new words that could fill the role of that given word. Secondly, they take into account n-gram approaches beyond unigram ( $n = 1$ ), bigram ( $n = 2$ ) or even trigram (the  $n$  typically used by researchers) up to an  $n$  of 5.

The math for n-gram models is given as:

$$\hat{P}(w_t | w_1^{t-1}) \approx \hat{P}(w_t | w_{t-n+1}^{t-1}).$$

This formula is used to construct conditional probability tables for the next word to be predicted. When modeling NLP, the odds in the fight against dimensionality can be improved by taking advantage of word order, and by recognizing that temporally closer words in the word sequence are statistically more dependent. What does this ultimately mean in the context of what has been discussed? What problem is this solving? The language model proposed makes dimensionality less of a curse and more of an inconvenience. That is to say, computational and memory complexity scale up in a linear fashion, not exponentially. It improves upon past efforts by learning a feature vector for each word to represent similarity and also learning a probability function for how words connect via a neural network. Let's take a closer look at said neural network.





We're presented here with something known as a Multi-Layer Perceptron. **What are those layers?** Three input nodes make up the foundation at the bottom, fed by the index for the word in the context of the text under study. The layer in the middle labeled tanh represents the hidden layer. Tanh, an activation function known as the hyperbolic tangent, is sigmoidal (s-shaped) and helps reduce the chance of the model getting “stuck” when assigning values to the language being processed. **How is this?** In the system this research team sets up, strongly negative values get assigned values very close to -1 and vice versa for positive ones. Only zero-valued inputs are mapped to near-zero outputs. The uppermost layer is the output — the softmax function. It is used to bring our range of values into the probabilistic realm (in the interval from 0 to 1, in which all vector components sum up to 1).

Don't overlook the dotted green lines connecting the inputs directly to outputs, either. The optional inclusion of this feature is brought up in the results section of the paper. It provides an interesting trade-off: including the direct connections between input and output causes the the

training time to be cut in half (10 epochs to converge instead of 20). Without them, the model produced better generalizations via a tighter bottleneck formed in the hidden layer.

It's possible for a sentence to obtain a high probability (even if the model has never encountered it before) if the words contained therein are similar to those in a previously observed one. There's the rub: Noam Chomsky and subsequent linguists are subject to criticisms of having developed too brittle of a system. This method sets the stage for a new kind of learning, deep learning. When utilized in conjunction with vector semantics, this is powerful stuff indeed. Through this paper, the Bengio team opened the door to the future and helped usher in a new era. An era of AI.

### **Perception**

Perception in [Artificial Intelligence](#) is the process of interpreting vision, sounds, smell, and touch. Perception helps to build machines or robots that react like humans. Perception is a process to interpret, acquire, select, and then organize the sensory information from the physical world to make actions like humans. The main difference between AI and robot is that the robot makes actions in the real world.

\*\*\*\*\*

