ARBA MINCH UNIVERSITY

INSTITUTE OF TECHNOLOGY

Faculty of Computing and Software Engineering

Introduction to Artificial Intelligence Course Module

C- CODE: COSC-5192

For Bachelor of Science Degree in Software Engineering

Compiled by: Melaku Meskele

March, 2023

Arba Minch, Ethiopia

Table of Contents

Course Goals

At the end of this course the students will be able to:

- Understand reasoning, knowledge representation and learning techniques of artificial intelligence
- Evaluate the strengths and weaknesses of these techniques and their applicability to different tasks
- Assess the role of AI in gaining insight into intelligence and perception
- Know classical examples of artificial intelligence
- Know characteristics of programs that can be considered "intelligent"
- Understand the use of heuristics in search problems and games
- Know a variety of ways to represent and retrieve knowledge and information
- Know the fundamentals of artificial intelligence programming techniques in a modern programming language
- Consider ideas and issues associated with social technical, and ethical uses of machines that involve artificial intelligence

CHAPTER 1

Introduction to Artificial Intelligence

1.1 Overview of Artificial Intelligence

What is Artificial Intelligence?

AI is the science and engineering of making intelligent machines, especially computer programs that are capable of simulating human intelligent behavior.

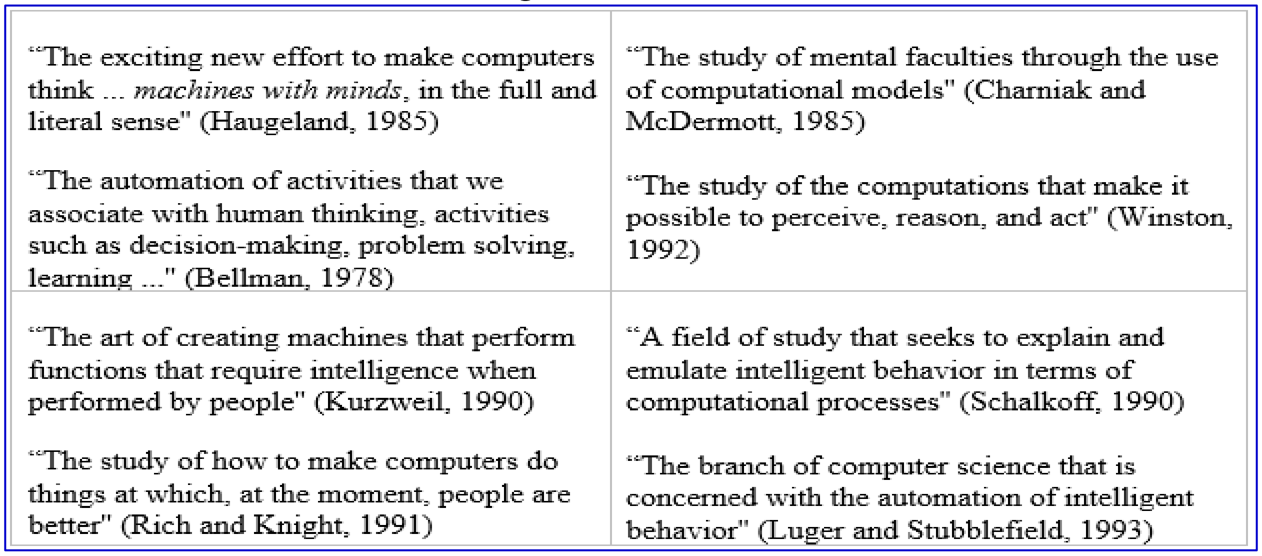
What is intelligence then?

- ❖ **Intelligence is the mental quality:** The computational part of the ability to achieve goals in the world.
- ❖ **Intelligence** includes the capacity for logic, learning, reasoning, critical thinking, creativity, perception, language understanding, problem solving, etc.

Artificial Intelligence is the branch of computer science dedicated to the development of computer systems or machines capable of performing tasks that typically require human intelligence. Examples of these tasks are: -

- ❖ Speech recognition
- ❖ Language Translation
- ❖ Performing complex surgical operation
- ❖ Driving a car in traffic (crowded environment)
- ❖ Proving a mathematical theorem

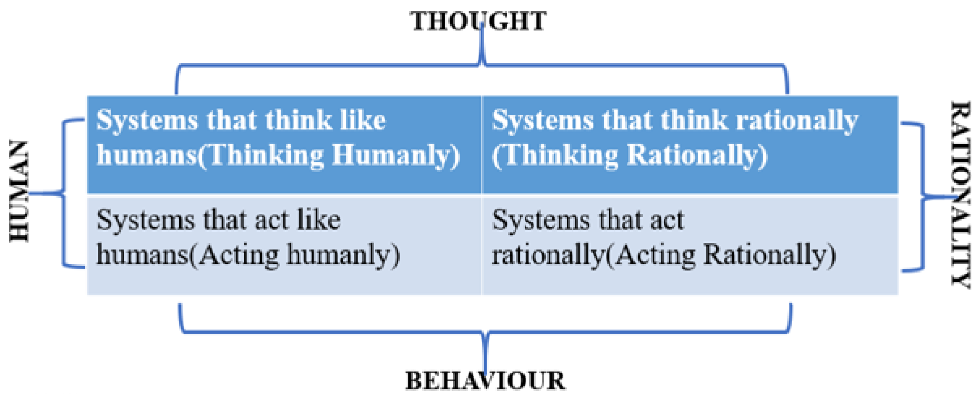
Another definitions of Artificial Intelligence



"The exciting new effort to make computers think ... *machines with minds*, in the full and literal sense" (Haugeland, 1985) "The automation of activities that we associate with human thinking, activities such as decision-making, problem solving, learning ..." (Bellman, 1978)	"The study of mental faculties through the use of computational models" (Charniak and McDermott, 1985) "The study of the computations that make it possible to perceive, reason, and act" (Winston, 1992)
"The art of creating machines that perform functions that require intelligence when performed by people" (Kurzweil, 1990) "The study of how to make computers do things at which, at the moment, people are better" (Rich and Knight, 1991)	"A field of study that seeks to explain and emulate intelligent behavior in terms of computational processes" (Schalkoff, 1990) "The branch of computer science that is concerned with the automation of intelligent behavior" (Luger and Stubblefield, 1993)

1.2 Approaches to AI

Views of AI fall into four categories:



Systems that think like humans(Thinking Humanly)	**Systems that think rationally (Thinking Rationally)**
Systems that act like humans(Acting humanly)	Systems that act rationally(Acting Rationally)

1.2.1 Acting Humanly: The Turing Test

- Alan Turing (1950) published "Computing Machinery and Intelligence".
- “Can machines think?" → "Can machines behave intelligently? “.
- He has proposed an operational test for intelligent behavior called the **Imitation Game.**
- ***Approaches of Turing Tests: -***
  - Interrogator asks the question to both computer & human via teletype, and receives the answer from both of them.
  - All three entities sit in an isolated location.
  - The Computer passes the test if the interrogator cannot distinguish the answers responded by the computer and humans.

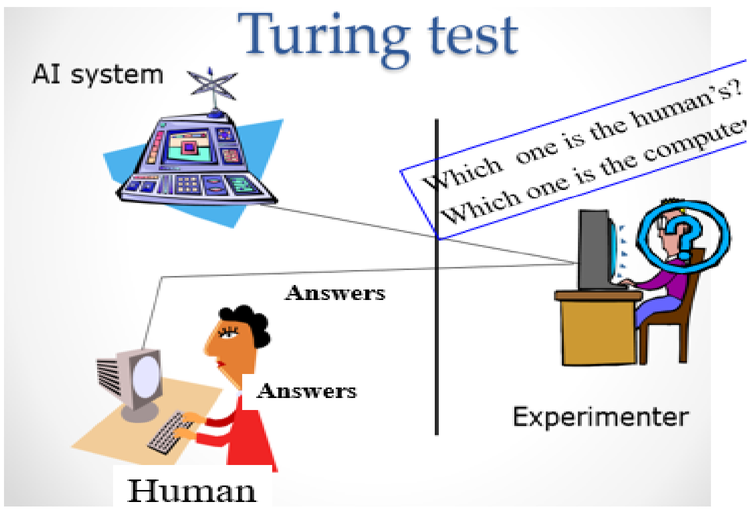


Figure 1.1: classical Turing test.

Captcha: *Completely Automated Public Turing test to tell Computers and Humans Apart*

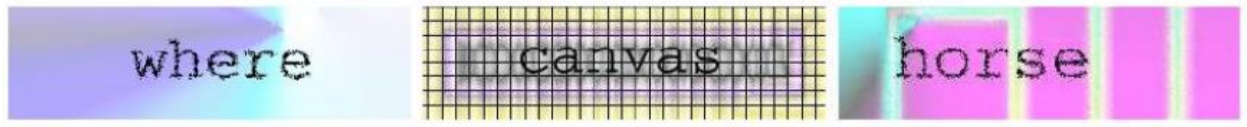


Figure 1.2: **Total Turing Test**

To pass the **classical Turing Test** computer needs to have the following capabilities **(Skills Required)**

- ➢ Natural language processing
- ➢ Knowledge representation
- ➢ Automated reasoning
- ➢ Machine learning.

► To pass the total Turing Test, the computer will need

- ➢ Computer vision
- ➢ Robotics

1.2.2 Thinking Humanly: Cognitive Modeling

When we say a given program thinks like a human, we must have some way of determining "how humans think?". We need to get *inside* the actual workings of human minds. There are two ways to do this: through **introspection--**trying to catch our thoughts as they go by--or through **psychological** experiments.

- ➢ This requires **knowledge of brain functions** (internal activities of the brain).
- ➢ **Cognitive Science** brings together computer models from AI and experimental techniques from psychology to construct the working of the human mind.

1.2.3 Thinking Rationally: "Laws of Thought"

Aristotle attempted this: What are correct arguments/thought processes? Several Greek schools developed various forms of ***logic notation and rules of derivation for thoughts.*** These laws of thought were supposed to govern the operation of the mind and provided the foundation of much of Artificial intelligence.

- **Problems with this approach;**
    - It is not easy, to take **informal knowledge** and state it in formal logical notation.
    - There is a big difference between being able to solve a problem **"in principle" and "doing so in practice"**.
    - Not all intelligent behavior is controlled/mediated by logic.

1.2.4 Acting Rationally: Rational Agent

Rational behavior: doing the right thing.

- **The right thing**: that which is expected to maximize goal achievement, given the available information**.**
- Doesn't necessarily involve thinking—e.g., blinking reflex—but thinking should be in the service of rational action.

For any given class of environments and tasks, we seek the agent (or class of agents) with the best performance. The rational agent should select an action that is expected to maximize its performance measure, given the percept sequence.

1.3 Suggested Components or Subfields of AI

Intelligent systems may be expected to involve the following components to interact with their environment.

1. ***Natural Language processing***
    - Enable computers to communicate in human language e.g., English, Amharic, ..
    - Speech recognition, understanding and synthesis
2. ***Knowledge representation***
    - Schemes to store information effectively & efficiently, both facts and inferences
3. ***Automated reasoning***
    - To use the stored information to answer questions and to draw new conclusions
4. ***Machine learning***
    - To adapt to new circumstances and to detect and extrapolate patterns

5. ***Computer vision***
   - To perceive and recognize objects based on patterns in the same way as the human visual system does
6. ***Robotics***
   - Enable computers to take actions or modify their environment

1.4 Goals of AI

- To replicate human intelligence (implement human intelligence on the Machine)
- To solve a knowledge-intensive task
- To build a machine tha can perform a task that requires human intelligence *ex., playing chess games, driving a car in traffic, proving a theorem, surgical operation*
- To Create Expert Systems − The systems which exhibit intelligent behaviour, learn, demonstrate, explain, and advice users.
- To develop helping machines find solutions to complex problems and to take over dangerous tasks from the human.

1.5 Foundations of Artificial Intelligence

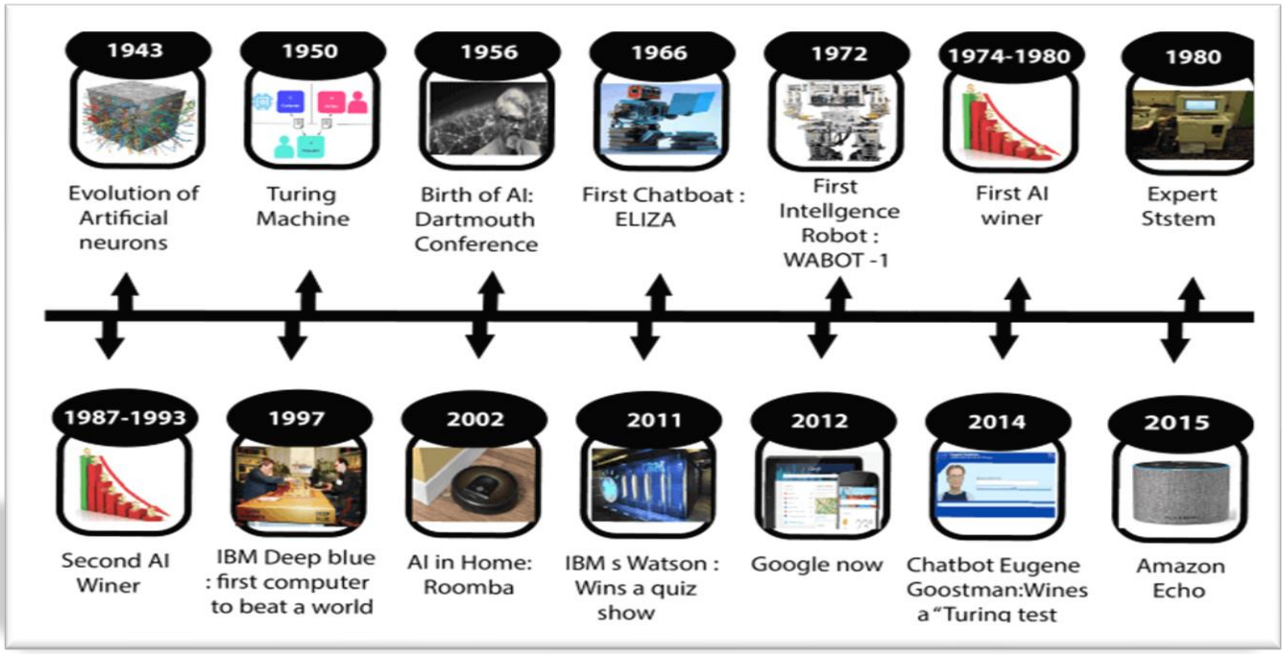
Artificial Intelligence has identifiable roots in several disciplines, particularly:

- **Philosophy**
  - Understanding the connection between how peoples think (connection b/n idea) and explicate that connection in a reasoned and a logic way.
  - Logic, methods of reasoning (Aristotle developed laws for reasoning)
- **Mathematics**
  - Boole introduced formal language for making logical inferences, knowledge representation (for instance, FOL) and proof, computation, (un)decidability, and (in)tractability.
- **Neuroscience** (Brain science)
  - Study how brains process information.
  - It enables to build powerful AI systems by simulating biological neural networks.
- **Economics** utility, decision theory, rational agents maximizing their own well-being (payoff)

- **Psychology /Cognitive science**
  - Study of human minds and their behaviors.
  - Understanding, how the human mind works/functions, and leads to intelligent behavior. ***Cognitive psychology initiated***
- **Computer science and Engineering**
  - Building fast and efficient computers.
  - Provides the artifact that makes AI application possible.
  - Design efficient algorithms and graphics
- **Control theory**
  - Design systems that maximize an objective function over time
- **Linguistics**
  - Natural languages processing
  - knowledge representation, grammar

1.6 Brief History of Artificial Intelligence

The following are some of the **milestones** in the history of AI which define the journey from the AI generation to till date development. How has AI is progressed /evolved at a different year?


1943
Evolution of Artificial neurons
1950
Turing Machine
1956
Birth of AI: Dartmouth Conference
1966
First Chatboat : ELIZA
1972
First Intellgence Robot : WABOT -1
1974-1980
First AI winer
1980
Expert Ststem
1987-1993
Second AI Winer
1997
IBM Deep blue : first computer to beat a world
2002
AI in Home: Roomba
2011
IBM s Watson : Wins a quiz show
2012
Google now
2014
Chatbot Eugene Goostman:Wines a "Turing test
2015
Amazon Echo

A. Maturation of Artificial Intelligence (1943-1952)

- ✓ **Warren McCulloch & Walter Pitts (1943):** proposed a model of artificial neurons.
- ✓ **Donald Hebb (1949):** demonstrated an **updating rule** for modifying the connection strength between **neurons**. His rule is called **Hebbian learning.**
- ✓ **Alan Turing (1950):** He pioneered **Machine learning** & published “Computing Machinery & Intelligence” in which he proposed tests (i.e. can machines think?).

B. The birth of Artificial Intelligence (1952-1956)

- ✓ **Allen Newell and Herbert A. Simon (1955):** created the first AI program, named **Logic Theorist**. This program proved 38 of 52 mathematics theorems.
- ✓ **John McCarthy(1956) :** The word "**Artificial Intelligence**" was first time adopted and called as **academic field**. Also, high-level computer languages such as FORTRAN, LISP, or COBOL were invented at that time.

C. Golden Years-Early enthusiasm (1956-1974)

- ✓ **Joseph Weizenbaum(1966):** created the first **chatbot** which is called **ELIZA**.
- ✓ **In the year 1972:** The first intelligent humanoid robot was built in Japan which was named **WABOT-1**.

D. The first AI winter (1974-1980)

- ✓ **Because** computer scientists dealt with a severe shortage of **funding** from the government for AI research.
- ✓ Publicity interest in AI was decreased.

E. A boom of AI (1980-1987)

- ✓ After AI winter duration AI came back with "Expert System".

F. The Second AI winter (1987-1993)

- ✓ Again, investors and the government stopped **funding** AI research due to high costs and its poor results.

G. The Emergence of Intelligent Agents (1993-2011)

- ✓ **In the year 1997, IBM Deep Blue** beats world chess champion, Gary Kasparov, and became the first computer to beat a world chess champion.
- ✓ **In the year 2002,** AI entered the home in the form of **Roomba, a vacuum cleaner**.

- ✓ **In the year 2006:** AI came into the **Business world**. Companies like Facebook, Twitter, and Netflix also started using AI.

H. Deep Learning, Big Data and Artificial General Intelligence (2011-present)

- ✓ **In the year 2006: IBM's Watson** won Jeopardy, a quiz show. **Watson** had proved that it could understand natural language and can solve tricky questions quickly.
- ✓ **In the year (2012):** Google has launched an Android app feature "**Google Now** “. It is one of the voice activated Google predictive search apps.
- ✓ **In the year (2014):** Chatbot "**Eugene Goostman**" won a competition in the infamous "Turing test."
- ✓ **In the year (2018):** The "**Project Debater**" from IBM debated on complex topics with two master debaters and also performed extremely well.

1.7 State of the Arts in AI

Intelligent Systems in our Everyday Life:

- ✓ **Banks**: automatic check readers, signature verification systems, automated loan application classification, detect fraudulent credit card transactions
- ✓ **Autonomous planning & scheduling:** NASA's on-board program controlled the operations of a spacecraft a hundred million miles from Earth.
- ✓ Self-Driving Cars
- ✓ Navigation Systems
- ✓ Email: Spam filtering
- ✓ Chatbots
- ✓ Medical Diagnosis
- ✓ Image recognition
- ✓ Virtual assistants: Alexa, Siri, Google Assistant, etc.

1.8 Application of Artificial Intelligence

AI has been dominant in various fields such as −

- ► **Gaming** − AI plays crucial role in the strategic games such as chess, poker, etc., where the machine can think of a large number of possible positions based on heuristic knowledge.
- ► **Natural Language Processing** −concerned with giving computers the ability to understand text and spoken words in much the same way human being can. It strives to

build a machine that understands and responds to the text or voice data and responds with text or speech of their own in much the same way a human do. Some NLP tasks include Machine Translation, Spell Checking, Grammar checking, text summarization, named entity recognition, word sense disambiguation, and speech recognition (speech-to-text).

- **Expert Systems** – The system that exhibits intelligent behaviour. It can provide explanations and advice to the users. Ex. Doctors use clinical expert systems to diagnose the patient.
- **Computer Vision** – Enables the computer to interpret, understand, and comprehend visual input (i.e. digital image, video) and drive useful information. Computers can accurately identify and classify objects and then react to what they see.
- **Healthcare**: Healthcare industries are applying AI to make a better and faster diagnosis than humans.
- **Social media:** social media sites such as Facebook, Twitter, and Snapchat contain millions of user profiles, which need to be stored and managed in a very efficient way. AI can organize and manage massive amounts of data. AI can analyze lots of data to identify the latest trends, hashtags, and requirements of different users.

CHAPTER 2

Intelligent Agents

At the end of this chapter students must be able to understand the following concepts:

- ❖ Overviews of an agent
- ❖ Agent and Environment
- ❖ Structure of agents
- ❖ Intelligent and Rational agent
- ❖ Types of intelligent agents

2.1 Introduction to Agents and Environment

An **agent** is anything that can be viewed as **perceiving** its environment through **sensors** and **acting** upon that environment through **actuators**. *Examples: Human agent; Robotic agent; Software agent.*

- ❖ A **human agent** has eyes, ears, and other organs for **sensors**, and hands, legs, mouth, and other body parts for **effectors**.
- ❖ A **robotic agent** substitutes the camera, infrared range finders, bump and …etc. for the sensors and various motors, grippers manipulators for the effectors.

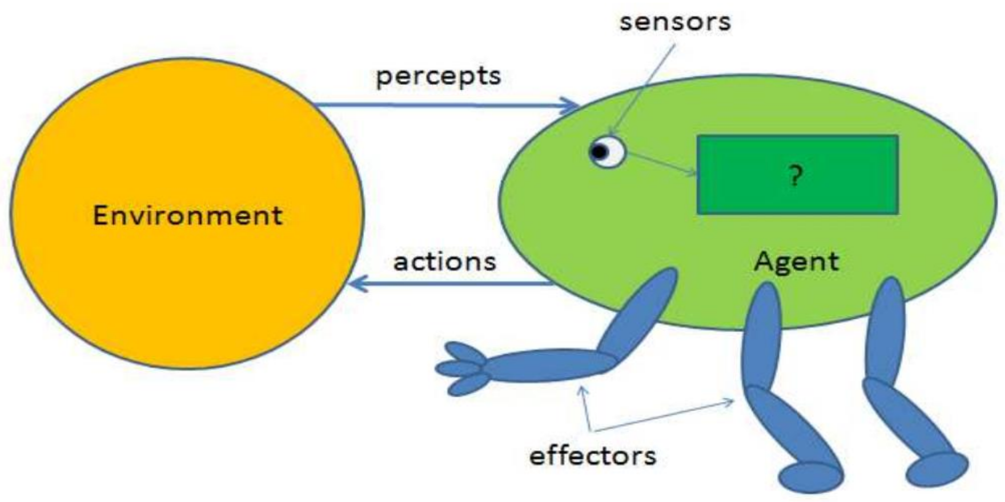


Figure 2.1: Agents interact with environments through sensors and actuators.

- ❖ **Percept** refers to the **agent's perceptual inputs**.
- ❖ **Percept Sequence** is the **complete history** of everything that the agent has perceived.

2.2 Rationality and Knowledge

- A rational agent "**does the right thing**".
  - *The right action is the one that will cause the agent to be most successful.*
- **Definition:**
  - *For each possible percept sequence, a **rational agent** should select an **action** that is expected to maximize its **performance measure**, given the evidence provided by the **percept sequence** and whatever built-in **knowledge** the agent has.*
- Rationality can be depending upon: -
  - *The performance measure defines the criterion of success.*
  - *The agent's prior knowledge of the environment.*
  - *The actions that the agent can perform.*
  - *The agent's percept sequence to date.*

2.3 Task Environment

To design a rational agent, we must specify the task environment. Specifying the task environment is always the first step in designing an agent.

- PEAS: **to specify a task environment**
  - **P**erformance measure
  - **E**nvironment
  - **A**ctuators
  - **S**ensors
- **Performance measure:** is the success criteria, which determines how successful an agent is.

Example: Specifying *PEAS* for an automated taxi driver

- ***P*erformance measure**:
  - Safe, fast, legal (minimize violation of traffic laws and other protocols), comfortable trip, maximize profits.
- **Environment**:
  - roads, other traffic/vehicles, pedestrians, customers, potholes, traffic signs
- ***A*ctuators**:
  - steering, accelerator, brake, signal, horn (sounding a warning), display

- *S***ensors**:
  - Cameras sonar and infrared, accelerometer), speedometer, GPS, odometer, engine-fuel-electrical system sensor, keyboard or microphone.

Activity 2.1

For each of the following activities, give a PEAS description of the task environment and characterize it in terms of the properties of task environments

a) Medical diagnosis system
b) Interactive AI tutor
c) Credit card fraud detection
d) Robot soccer player
e) Part parking robot
f) Image Analysis

2.4 Properties of Task Environments

1. Fully observable Vs Partially observable

Fully Observable: The agent's sensors give it access to the complete state of the environment at each point of time. Fully observable environments are **convenient** because the agent does not need to maintain any **internal state** to keep track history of the world.

- ***Example****: Chess game,* Cross Word

Partially observable: Parts of the environment are inaccessible (parts of the state are simply missing from the sensor data). Agent must make informed guesses about the world.

- **Example**: Self-driving car, Poker

2. Single vs. multi-agent

If only **one agent** is involved in an environment and **operating by itself** then such an environment is called a single-agent environment. If **multiple agents** are operating in an environment then such environment is called a multi-agent environment.

- For example, an agent solving a **crossword puzzle** by itself is clearly in a single-agent environment, whereas an agent **playing chess** is in a two-agent environment.
- Competitive vs. cooperative, chess game vs Self-driver

*3. **Static vs. dynamic***

Dynamic environment: the environment can change while the agent is deliberating over what to do. Static environments don't change. Static environments are easy to deal with because the agent does not need to keep looking at the world while it is deciding on an action. However, in dynamic environment agents need to keep looking at the world while it is deciding on an action.

- **Tax driving** is an example of a dynamic environment whereas crossword, Poker are an example of a static environment.

4. Episodic vs. Sequential

In an episodic task environment, the agent's experience is divided into atomic episodes. Choice of action in each episode depends only on the episode itself. That means an agent performs **independent task** in each episode. The agent's current decision doesn't affect future decisions. Many classification tasks are **episodic**.

- For example, an agent that has to **spot defective products** on an assembly line bases each decision on the current part, regardless of previous decisions.

In sequential environments, the agent operates in a series of connected episode. The agents current choice/decision will affect future decisions.

- **Example:** Chess game**,** Cross-word, Poker, and taxi driver are sequential: in both cases, short-term actions can have long-term consequences.

5. Deterministic vs. Stochastic

Deterministic environment: an agent's current state and selected action can determine the next state of the environment.

- **Example**: Cross Word, Chess game is an example of deterministic

A stochastic environment is **random** in nature and cannot be determined completely by an agent.

- **Example:** Taxi driving is clearly stochastic in this sense, because one can never predict the behavior of traffic exactly.

6. Discrete vs. Continuous

In a discrete Environment there are a limited number of distinct, clearly defined percepts and actions. At any given state there are only finitely many actions to choose from.

- For example, the **chess** comes under a discrete environment because it has a finite number of distinct states, and a discrete set of percepts and actions.
- **Taxi driving** is a continuous-state and continuous-time problem: the speed and location of the taxi is continuous values.

2.5 Structure of Agents

The task of AI is to design an **agent program** that implements the agent function. The structure of an intelligent agent can be viewed as:

Agent = Architecture + Agent Program

- **Architecture** → the machinery/computing device (e.g. PC, robotic car) with physical sensors and actuators that an agent program will run on.
- **Agent function** maps any given percept sequence to an action [f: P ∗ → A]
  - ✓ The agent function is an abstract mathematical description
  - ✓ Designer needs to construct a **table** that contains the appropriate action for every possible percept sequence
- **The agent program** runs on the physical architecture to produce f .
  - ✓ concrete implementation, running within some physical system

2.6 Intelligent Agents

An intelligent agent is an autonomous entity that acts upon an environment using sensors and actuators for achieving goals. An intelligent agent may learn from the environment to achieve their goals. An intelligent Agent:

- Must have the ability to perceive/sense its environment
- Must be able to make decision
- Must be able to take an action based on a decision taken
- Must be able to take rational action

2.7 Types of Intelligent Agents

Five basic agent types, arranged in order of increasing generality:

- Simple reflex agents
- Model-based reflex agents
- Goal-based agents
- Utility-based agents
- Learning agents

Each kind of agent program combines particular components in particular ways to generate actions. All these can be turned into learning agents.

2.6.1 Simple reflex Agents

These agents select an action based on the **current percept**, ignoring the rest of the percept history. It works on the ***condition-action rule***: -which means it directly maps the current percept to action. **For example**, the vacuum agent whose agent function is tabulated in Figure 2.3 is a simple reflex agent, because its decision is based only on the current location and on whether that location contains dirt. If location A is dirty." Then, this triggers some established connection in the agent program to the action "**Suck**".

❖ We call such a connection, **condition–action rule**, and written as follows.

Example 1: if location A is dirty then suck

Example 2: if car-in-front-is-braking then initiate-braking.

This agent only succeeds in the fully observable environment.

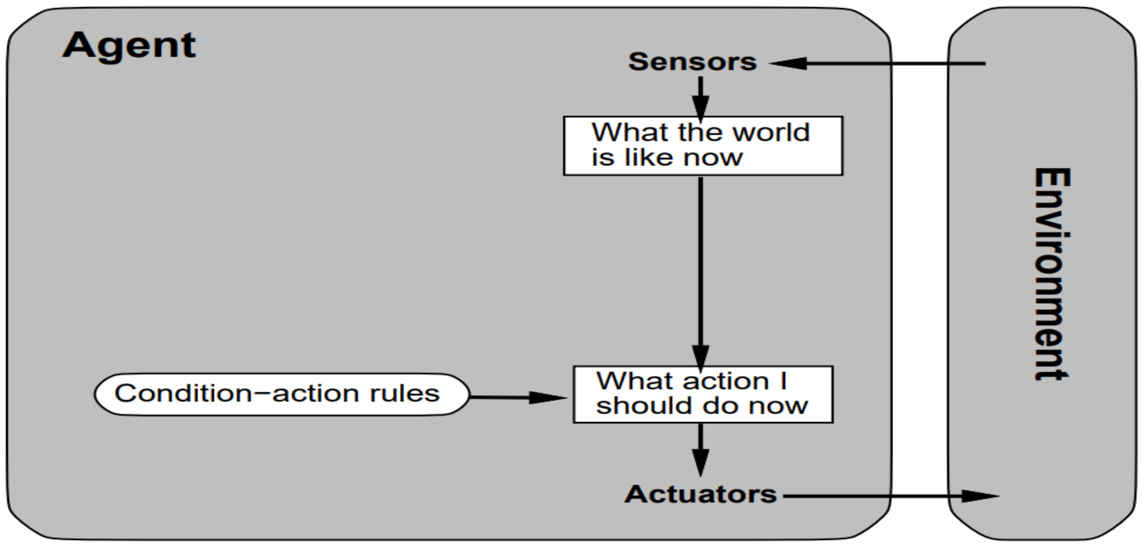
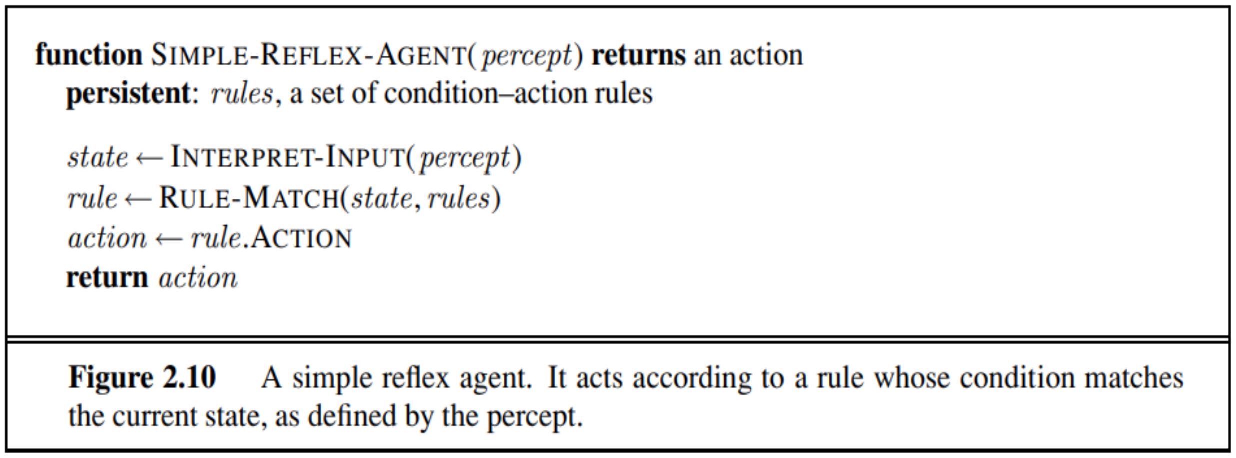


Figure 2.2: schematic diagram of simple reflex Agents



```
function SIMPLE-REFLEX-AGENT(percept) returns an action
    persistent: rules, a set of condition–action rules

    state ← INTERPRET-INPUT(percept)
    rule ← RULE-MATCH(state, rules)
    action ← rule.ACTION
    return action
```

Figure 2.10 A simple reflex agent. It acts according to a rule whose condition matches the current state, as defined by the percept.

- ✓ INTERPRET-INPUT function generates an abstracted description of the current state from the percept.
- ✓ RULE-MATCH function returns the first rule in the set of rules that matches the given state description.

2.6.2 Model-based reflex agents

The most effective way to handle partial observability is for the agent to keep track of the part of the world it can't see now. Model based agents maintains internal state or keeps track of percept history in order to reveal some of the unobservable aspects of the current state. For other driving tasks such as changing lanes, the agent needs to keep track of where the other cars are if it can't see them all at once. The agent combines the **current percept** with the **internal state** to generate the updated description of the **current state**.

Updating internal state information requires two kinds of knowledge to be encoded in the agent program.

- ❖ how the world evolves in-dependently from the agent.
- ❖ how the agent actions affects the world.

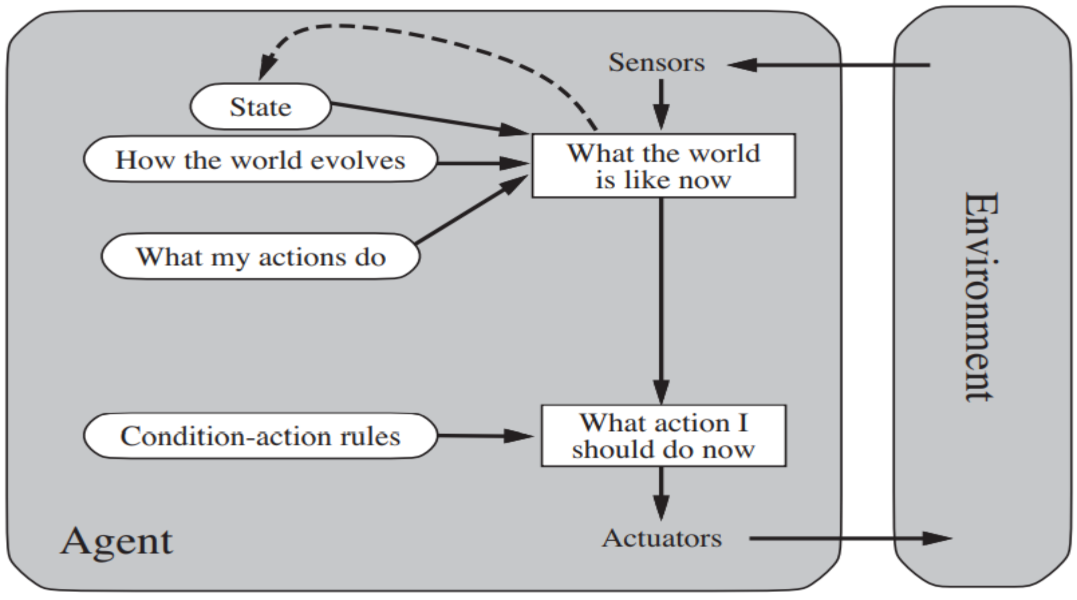


Figure 2.3: structure of the *Model based reflex agents:* showing how the current percept is combined with the old internal state to generate the updated description of the current state

```
function MODEL-BASED-REFLEX-AGENT(percept) returns an action
  persistent: state, the agent's current conception of the world state
              model, a description of how the next state depends on current state and action
              rules, a set of condition–action rules
              action, the most recent action, initially none

  state ← UPDATE-STATE(state, action, percept, model)
  rule ← RULE-MATCH(state, rules)
  action ← rule.ACTION
  return action
```

Figure 2.12 A model-based reflex agent. It keeps track of the current state of the world, using an internal model. It then chooses an action in the same way as the reflex agent.

2.6.3 Goal Based Agent

Knowing something about the current state of the environment is not always enough to decide what to do. For example, at a road junction, the taxi can turn left, turn right, or go straight on. The correct decision depends on where the taxi is trying to get to. In addition to current state description, the agent needs "goal information". The agent program combines "**goal information**" with the "**model".** This allows the agent to choose among **multiple possibilities**, select the one which reaches a goal state. Usually requires Searching and planning to finding action sequences that achieve the agent's goals. ***Example***.: *GPS system finding path to certain destination.*

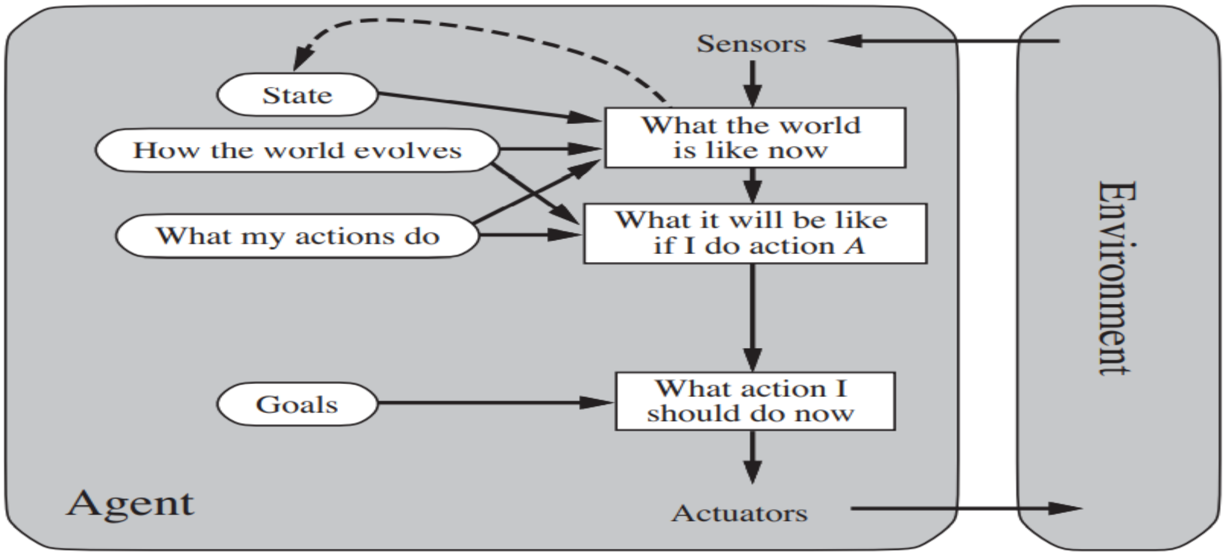

Sensors
State
How the world evolves
What my actions do
What the world is like now
What it will be like if I do action A
Goals
What action I should do now
Agent
Actuators
Environment

Figure 2.4: Goal based agents: *Decision making of this kind is fundamentally different from the condition– action rules, this involves consideration of the future—both "What will happen if I do such-and-such?". Will that make me happy?"*

2.6.4 Utility-based agents

Goals alone are not enough to generate **high-quality behavior** in most environments. For example, many action sequences will get the taxi to its destination (thereby achieving the goal) but some are quicker, safer, more reliable, or cheaper than others. **Utility functions** assign a score to any given sequence of environment states. Based on the utility, an agent chooses the actions that maximize its **expected utility**/performance measure/ for each state of the world. Example: Taxi driving agent can easily distinguish between more and less desirable ways of getting to its destination. ***The term utility, can be used to describe*** "how happy they would make the agent"

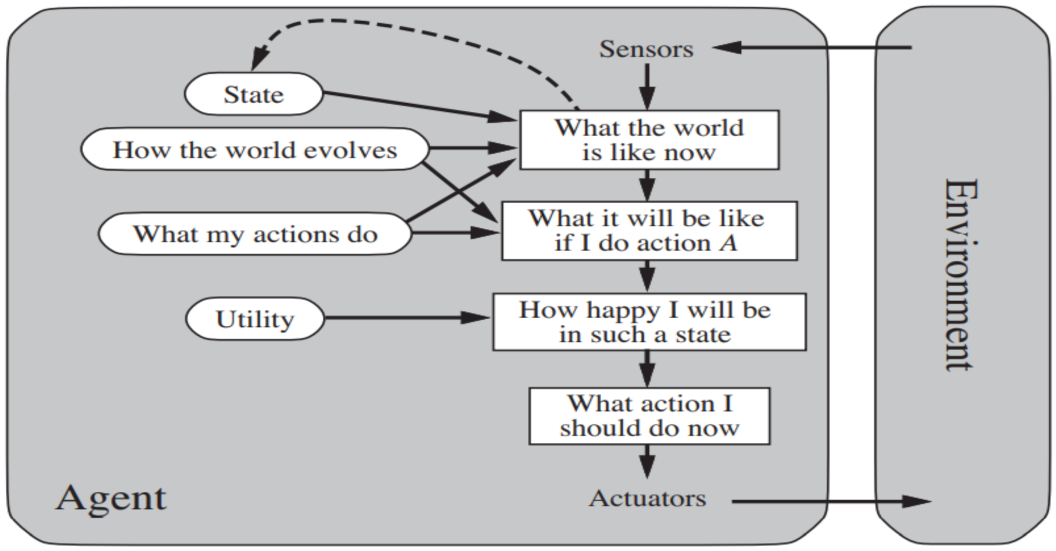

Sensors
State
How the world evolves
What my actions do
Utility
What the world is like now
What it will be like if I do action A
How happy I will be in such a state
What action I should do now
Agent
Actuators
Environment

Figure 2.5: Utility-based agents. It uses a model of the world, along with a utility function that measures its preferences among states of the world. Then it chooses the action that leads to the best expected utility.

2.6.5 Learning agents

Learning agent is capable of learning from its experience. It starts with basic knowledge and then able to act and adapt autonomously through learning, to improve its performance over time.

Autonomous agent: The autonomy of an agent is the extent to which its behavior is determined by its own experience (with the ability to learn and adapt).

A learning agent is divided into four conceptual components.

I. **Learning element**, which is responsible for making improvements. *It uses **feedback** from the **critic** on how the agent is doing and determines how the **performance element** should be modified to do better in the future.*

II. **Performance element**, which is responsible for selecting external actions. It is considered to be the entire agent: it takes in percepts and decides on actions.

III. **Critic** tells the learning element how well the agent is doing with respect to a fixed performance standard.

IV. **Problem generator**: It is responsible for suggesting actions that will lead to new and informative experiences.

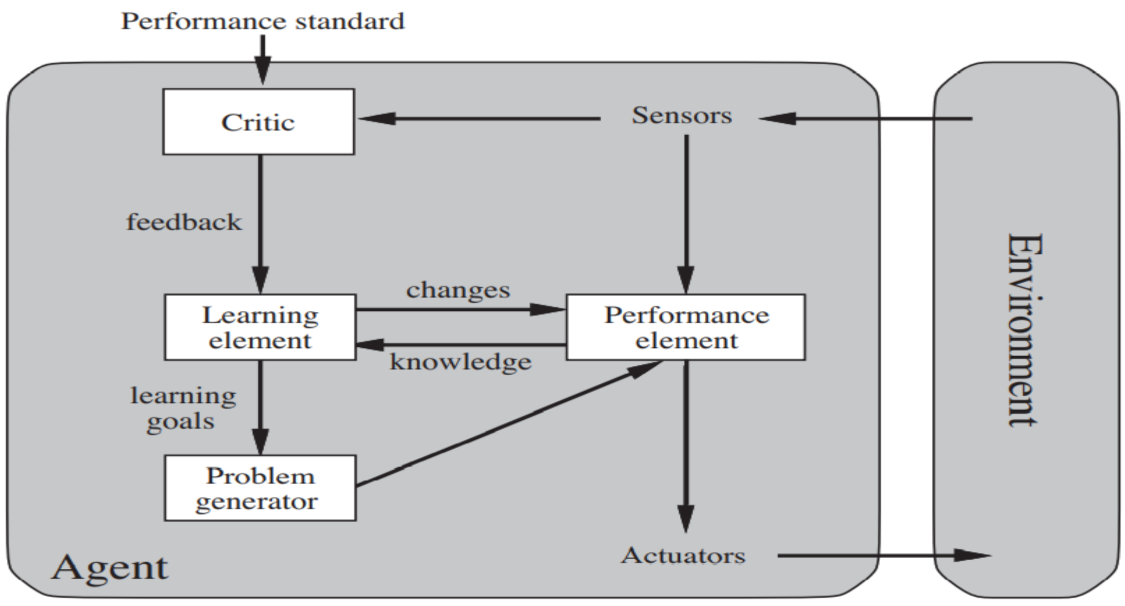


Figure 2.6: Learning agents

CHAPTER 3

Solving Problems by Searching & Constraint Satisfaction Problem

3.1 Problem-Solving by Searching

An important application of AI is **problem solving**. It is a process of generating a solution to a given problem. How an agent can find a **sequence of actions** that achieves its goals when no single action will do?" **Searching** is the process of looking for a sequence of actions that reaches the goal. The **search algorithm** takes a problem as input and returns a **solution** in the form of an action sequence.

3.2 Problem-solving agent

A goal-driven agent and focuses on satisfying the goal.

- Decide what to do by finding sequences of actions that lead to desirable states.
- To find the **sequence of actions** they use **search strategies**.

► What choices are the agents searching through?

- Problem solving: Action combinations (move 1, then move 3, then move 2...)
- Natural language: Ways to map words to parts of speech
- Computer vision: Ways to map features to object model
- Machine learning: Possible concepts that fit examples seen so far
- Route finding: Sequence of moves to reach goal destination

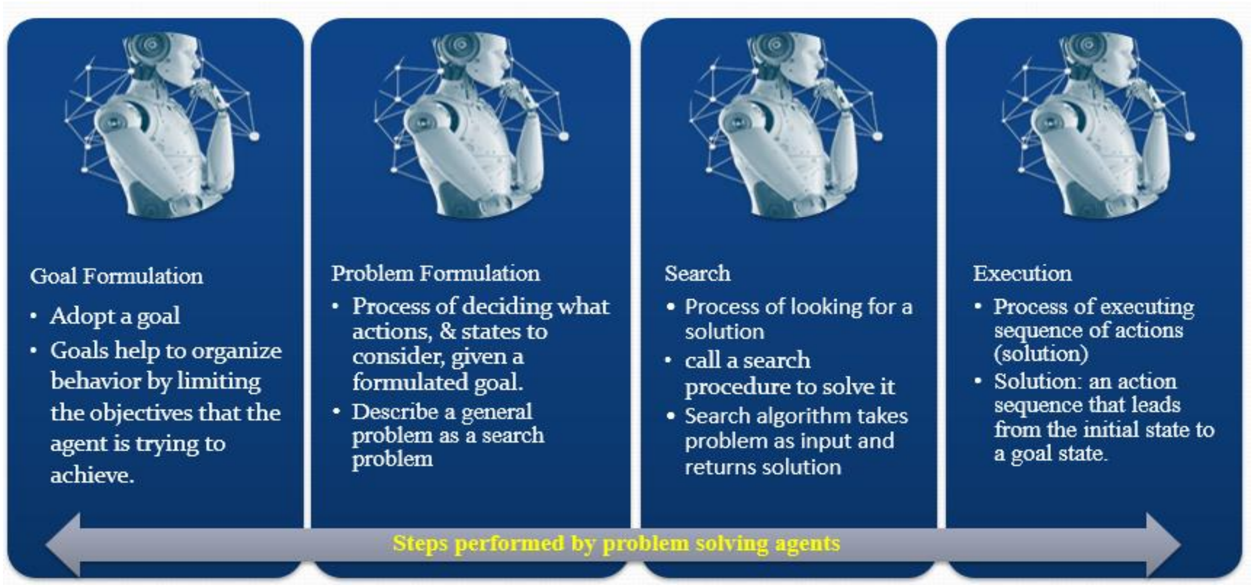

Goal Formulation
• Adopt a goal
• Goals help to organize behavior by limiting the objectives that the agent is trying to achieve.
Problem Formulation
• Process of deciding what actions, & states to consider, given a formulated goal.
• Describe a general problem as a search problem
Search
• Process of looking for a solution
• call a search procedure to solve it
• Search algorithm takes problem as input and returns solution
Execution
• Process of executing sequence of actions (solution)
• Solution: an action sequence that leads from the initial state to a goal state.
Steps performed by problem solving agents

Figure 3.1: Steps performed by problem solving agents

3.3 Well-defined problems and solutions

A problem can be defined formally by using five components:

- **Initial state:** the state where an agent starts in.
- **Action**: A description of the possible actions available to the agent
    - Given a particular state s, ACTIONS(s) returns the set of actions that can be executed in s.
- **Transition model:** A description of what each action does.
    - The function RESULT(s, a) returns the state that results from doing action **a** in state **s**. **Example**: RESULT(In(Arba Minch),Go(Hawassa)) = In(Hawassa) .
- **The goal test** determines whether a given state is a goal state.
    - Goal test can be: **explicit**, e.g., x = "at Bucharest“, implicit, e.g., Checkmate(x), NoDirt(x)
- **A path cost** function assigns a numeric cost to each path.
    - e.g., the sum of distances, number of actions executed, etc.
    - The step cost of taking action a in state s to reach state s’ is denoted by c(s, a, s’). The problem-solving agent chooses a cost function that reflects its own performance measure

The preceding elements define a problem and can be gathered into a single data structure that is given as input to a problem-solving algorithm. Solution is a sequence of actions that leads from the initial state to a goal state. Solution quality is measured by the path cost function, and an optimal solution has the lowest path cost among all solutions

Example : Holiday Planning

Imagine an agent in the city of Arad, Romania, enjoying a touring holiday. The agent’s performance measure contains many factors: it wants to improve its suntan, improve its Romanian, take in the sights, enjoy the nightlife (such as it is), avoid hangovers, and so on. Now, suppose the agent has a non-refundable ticket to fly out of Bucharest the following day. In that case, it makes sense for the agent to adopt the goal of getting to Bucharest.

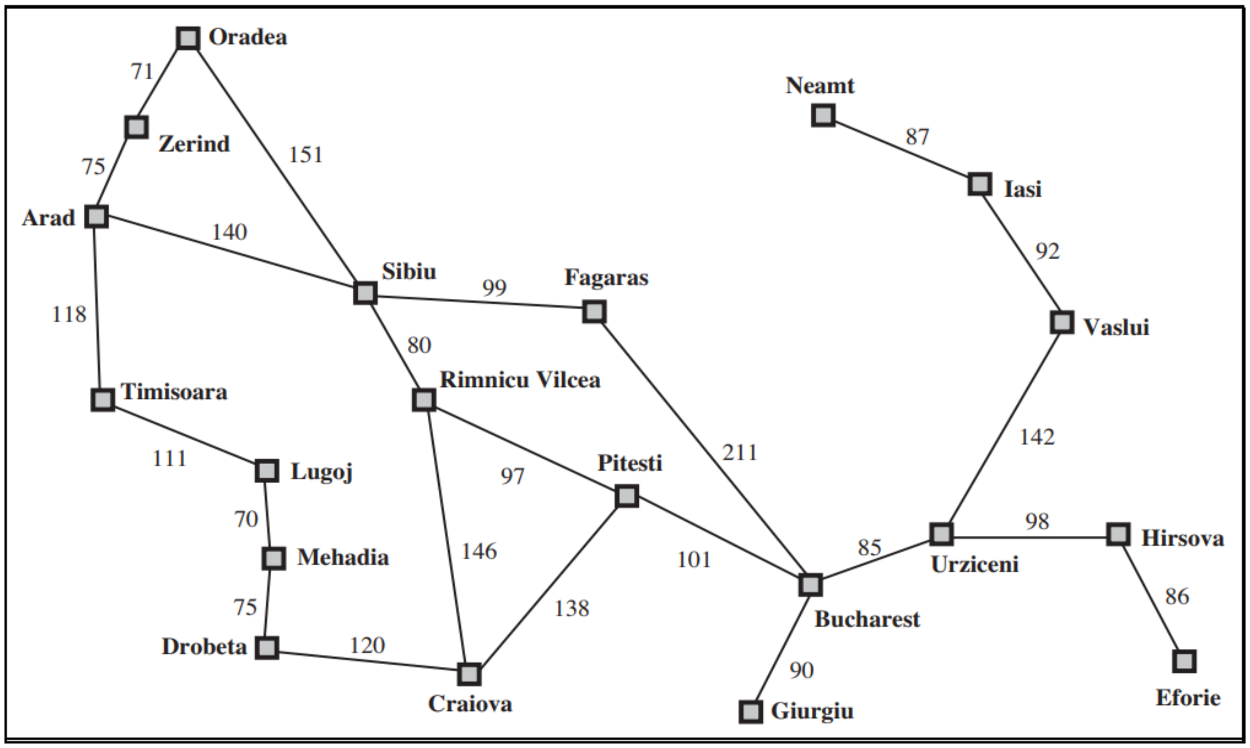


Figure 3.2: A simplified road map of part of Romania.

Formulate the goal and problem for the Romanian problem

- ► Formulate goal: be in Bucharest
- ► Formulate problem:
  - ❖ Initial state: IN(Arad)
  - ❖ Actions: From the state In(Arad), the applicable actions are {Go(Sibiu), Go(Timisoara), Go(Zerind)}. Shortly, IN(ARAD)->*GO(Zerind), GO(Siblu), GO(Timisoara)*
  - ❖ Transition models: Result(IN(Arad),GO(Zerind))➔ IN(Zerind)
  - ❖ Goal Test: IN(x)==IN(Bucharest)
  - ❖ Path Cost: C(IN(Arad), GO(Zerind), IN(Zerind)’) = 75

A simple problem-solving agent algorithm

```
function SIMPLE-PROBLEM-SOLVING-AGENT(percept) returns an action
   persistent: seq, an action sequence, initially empty
               state, some description of the current world state
               goal, a goal, initially null
               problem, a problem formulation

   state ← UPDATE-STATE(state, percept)
   if seq is empty then
       goal ← FORMULATE-GOAL(state)
       problem ← FORMULATE-PROBLEM(state, goal)
       seq ← SEARCH(problem)
       if seq = failure then return a null action
   action ← FIRST(seq)
   seq ← REST(seq)
   return action
```

3.4 Problem Abstraction

The real world is complex and has more details. Irrelevant details should be removed from state space and actions, which is called abstraction.

- ➢ What's the appropriate level of abstraction?
    - the abstraction is valid, if we can expand it into a solution in the more detailed world
    - the abstraction is useful, if carrying out the actions in the solution is easier than the original problem
    - remove as much detail as possible while retaining validity and ensuring that the abstract actions are easy to carry out.
- ➢ A key element of successful problem formulation relates to abstraction – the process of removing (inessential) details from a problem representation.
- ❖ **We must abstract the state description**
    - ➢ Compare the simple state description we have chosen, In(Arad), to the actual country trip, but the actual state of the world includes so many things: road condition, traveling companions, weather, etc.

- ➢ All these considerations are left out of our state descriptions because they are irrelevant to the problem of finding a route to Bucharest

❖ **We must abstract the actions**

- ➢ A driving action has many effects. But we omit many actions altogether e.g. turning on the radio, turn steering wheel right etc., Our formulation considers only the change in location.
- ➢ Each abstract action should be "easier" than the original problem!

❖ **Example 2: 8-puzzle**

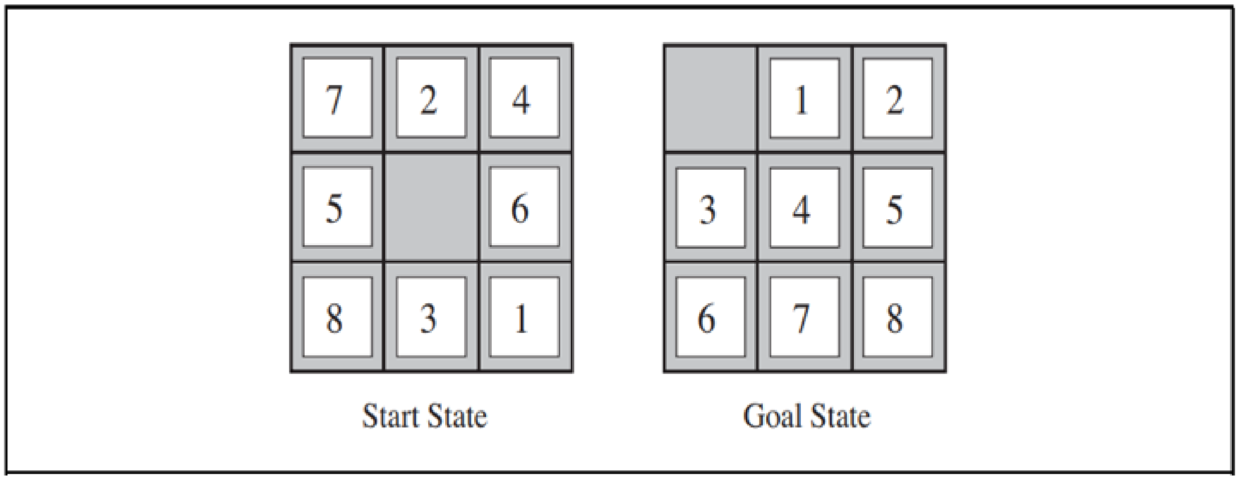


Figure 3.3: The typical instances of the 8-puzzle

- ❖ **States**: location of each of the eight tiles and the blank in one of the nine squares.
- ❖ **Initial state**: Any state can be designated as the initial state.
- ❖ **Actions**: specify actions as movements/sliding of the blank space **R, L, U, D**
- ❖ **Transition model:** Given a state and action, this returns the resulting state; for example, if we apply Left to the start state in Figure 3.3, the resulting state has the 5 and the blank switched.
- ❖ **Goal test:** This checks whether the state matches the goal configuration shown in Figure 3.3. (Other goal configurations are possible.)
- ❖ **Path cost:** Each step costs 1, so the path cost is the number of steps in the path.

Searching for Solutions

Having formulated some problems, we now need to solve them. A solution is an action sequence, so search algorithms work by considering various possible action sequences. The possible action sequences starting at the initial state form a search tree with the initial state at the root; the branches are actions and the nodes correspond to states in the state space of the problem.

The root node of the tree corresponds to the initial state, In(Arad). The first step is to test whether this is a goal state. Clearly, it is not, then we need to consider taking various actions. We do this by **expanding** the **current state**; thereby **generating** a new set of states. In this case, we **add three branches** from the **parent node** In(Arad) leading to three new **child nodes:** In(Sibiu), In(Timisoara), and In(Zerind). Now we must choose which of these three possibilities to consider further.

A node with no children is called a **leaf**; the set of all leaf nodes available for expansion at any given point is called the **frontier (**open list). Figure 3.4, the frontier of each tree consists of those nodes with bold outlines. The process of **expanding nodes** on the **frontier** continues until either a solution is found or there are no more states to expand. Tree search algorithms all share this basic structure; they vary primarily according to how they choose which state to expand next—the so-called **search strategy.**

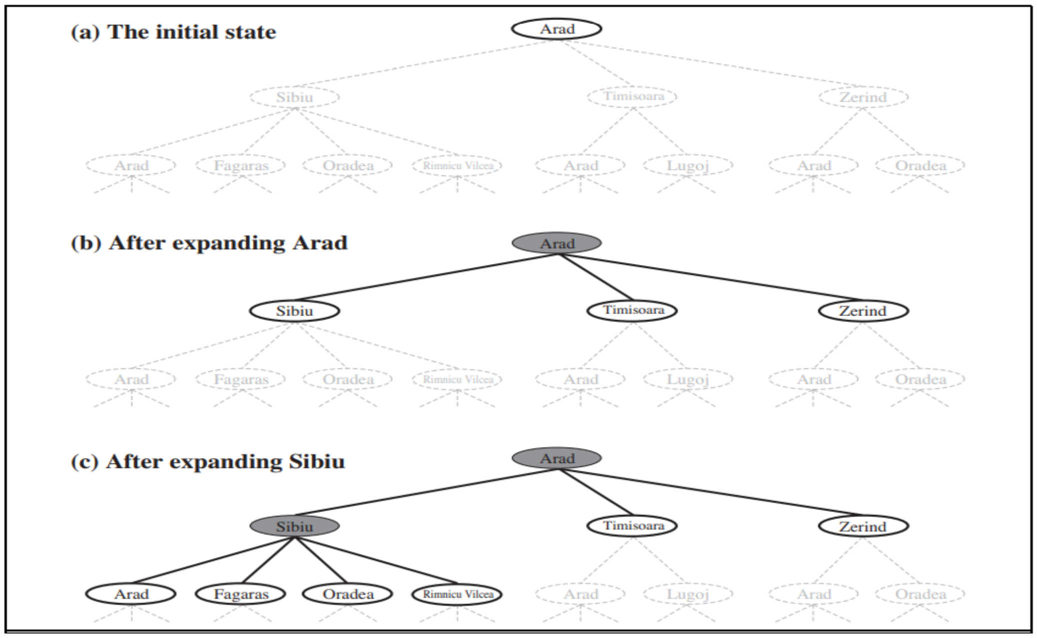


Figure 3.4 Partial search trees for finding a route from Arad to Bucharest. Nodes that have been expanded are shaded; nodes that have been generated but not yet expanded are outlined in bold; nodes that have not yet been generated are shown in faint dashed lines.

3.5 Avoiding repeated states

Naturally, if we allow for redundant paths, then a formerly tractable problem can become intractable" "Algorithms that forget their history are doomed to repeat it." To avoid exploring redundant paths we can augment the algorithm with a data structure called the **explored set** (also known as the **closed list**), which remembers every expanded node.

Newly generated nodes that match previously generated nodes—ones in the explored set or the frontier— (***we discard nodes in explored set instead of adding them to the frontier***). Path from **Arad** to **Sibiu** and back to **Arad** again! In (Arad) is a repeated state in the search tree, generated in this case by a loopy path. Search tree for Romania is infinite because there is no limit to how often one can traverse a loop.

3.6 Search Strategies

A search strategy is **defined by picking the order of node expansion**. There are 2 kinds of search, based on whether they use **information about the goal**.

- **Uninformed Search**
  - Does not use any domain knowledge (*Blind search*), which means that the strategies have no additional information about states beyond that data provided in the problem definition.
  - Uninformed search strategies use only the information available in the problem definition.
- **Informed Search**
  - Uses domain knowledge (*Heuristic search*)
  - Informed search, where the algorithm is given some guidance.

3.6.1 Performance easures of search algorithm

We can evaluate the search algorithm's performance in four ways:

- **Completeness**: Does it always find a solution when one exists?
- **Optimality**: Does the strategy find the optimal solution?
- **Time complexity**: How long does it take to find a solution (worst case)? number of nodes generated
- **Space complexity**: How much memory is needed to perform the search? maximum number of nodes in memory.

- Time and space complexity is measured in terms of
    - ✓ b: maximum branching factor of the search tree (maximum number of successors of any node)
    - ✓ d: depth of the least-cost solution (of the shallowest goal node)
    - ✓ m: maximum depth of the state space (may be ∞)
    - ✓ The size of the state space graph (|V|+|E|)

3.6.2 Uninformed Search Strategies

Uninformed strategies use only the information available in the problem definition

- Breadth-first search
- Depth-first search
- Depth-limited search
- Uniform-cost search
- Iterative deepening search

1. **Breadth-first Search (BFS)**

BFS is a simple search strategy in which the root node is expanded first, then all the successors of the root node are expanded next, then their successors, and so on. BFS is an instance of the general graph-search algorithm in which the shallowest unexpanded node is chosen for expansion. In general, all the nodes are expanded at a given depth in the search tree before any nodes at the next level are expanded. BFS is known as Level by level search strategy.

- Implementation:
    - fringe is a FIFO queue, i.e., new successors go at the end.
    - Thus, new nodes (which are always deeper than their parents) inserted into the end of the queue (go back of the queue), and old nodes, which are shallower than the new nodes, get expanded first.

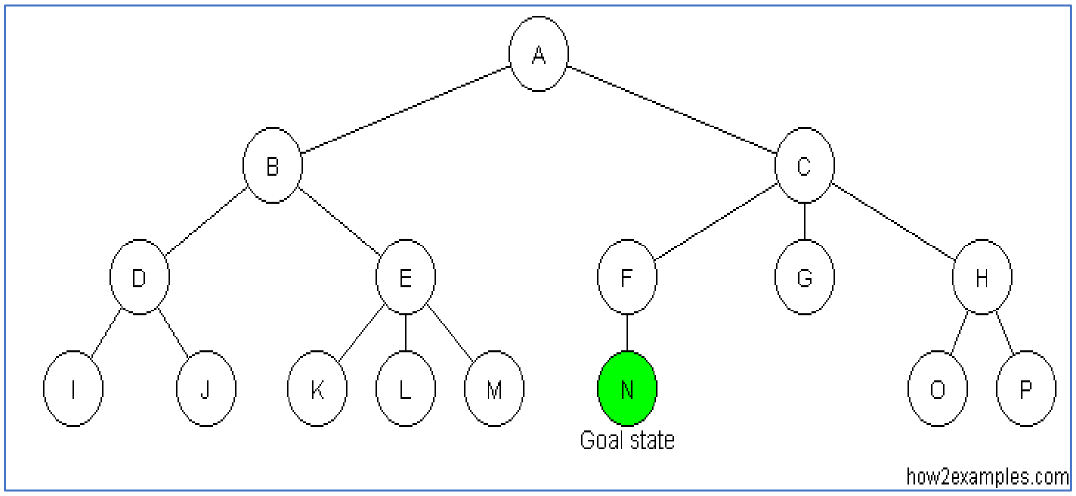


Figure 3.6: Search strategies of breadth first search algorithm

Evaluating Breadth First Search Algorithm

Time Complexity:

Imagine searching a **uniform tree** where every **state/node** has b successors. The root of the search tree generates **b** nodes at the **first level**, each of which generates b more nodes, for a total of b^2 at the **second level**. Each of these generates b more nodes, yielding b^3 nodes at the **third level**, and so on. Now suppose that the solution is at **depth/level** d. In the worst case, it is the last node generated at that level. Then the **total number of nodes generated** is

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

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

BFS consumes much time to reach the goal node for large instances, in case of, if the solution is far away from the root node.

Completeness:

- It is a complete strategy as it definitely finds the goal state **if b is finite**. Not optimal in general.

Optimality:

- Yes (if cost = 1 per step, identical for all). It gives an optimal solution if the path cost of each node is same.

Space Complexity:

- The space complexity of BFS is **O(b^d)**, keeps every node in memory
- It requires lots of memory because every expanded node has to stay in memory in the **explored set**.
- **Space** is the bigger problem (more than time)

2. Depth first Search (DFS)

DFS algorithm starts with the initial node, then goes to the deepest level of search tree, where the nodes have no successors (no children). DFS then backtracks from the dead-end towards the most recent node that is yet to be completely unexplored successors

Implementation: *frontier = LIFO Queue, i.e., put successors at front.*

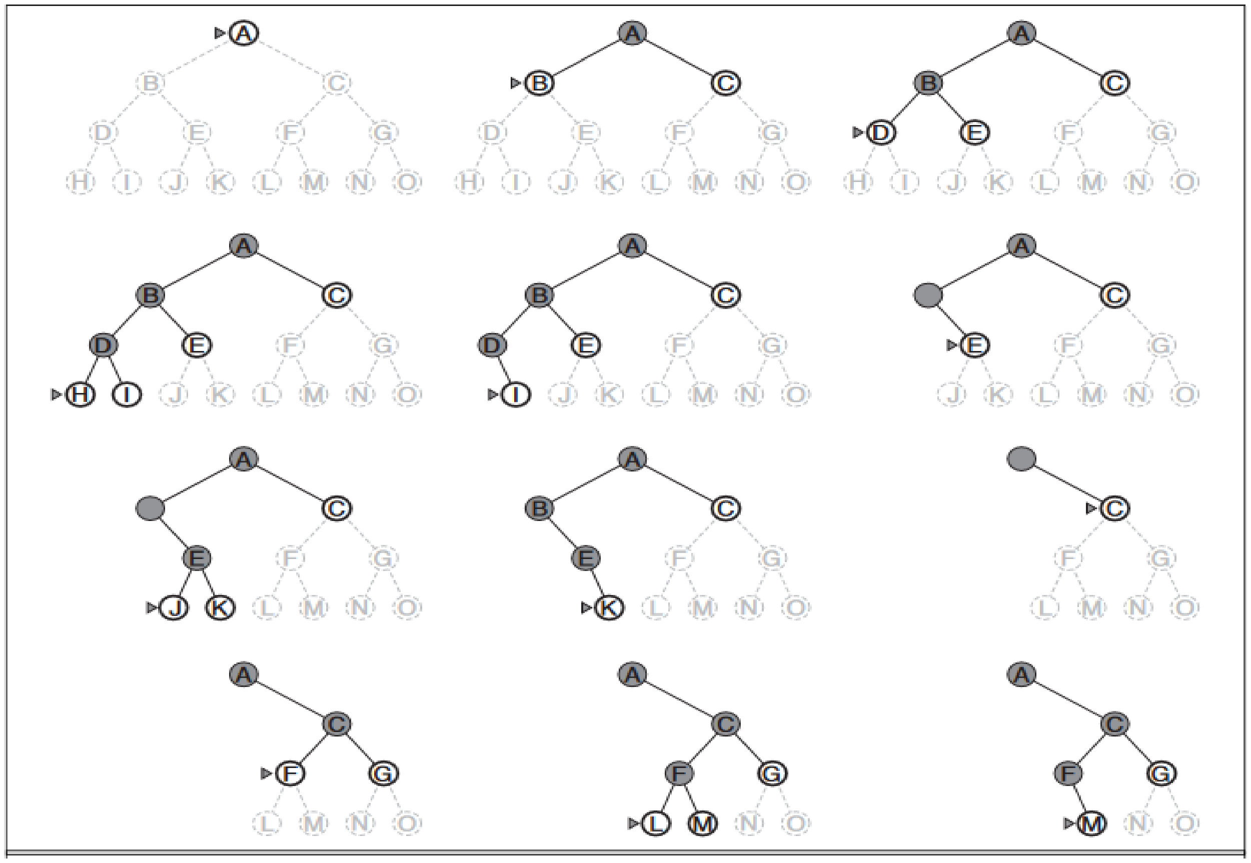


Figure 3.6 Depth-first search on a binary tree. The unexplored region is shown in light gray. Explored nodes with no descendants in the frontier are removed from memory. Nodes at depth 3 have no successors and M is the only goal node.

Evaluation of depth first search algorithm

Completeness:

- ► No: fails in infinite-depth spaces, or spaces with loops. complete in finite spaces
- ► DFS algorithm goes for deep down searching and sometime it may go to the infinite loop.

Optimality:

- ► DFS is **non-optimal**.
- ► For example, in Figure 3.16, DFS will explore the entire left subtree even if node C is a goal node. If node J were also a goal node, then DFS would return it as a solution instead of C, which would be a better solution; hence, DFS is not optimal.

Time complexity: Needs Exponential time, O(b^m)

- ❖ A depth-first tree search, may generate all of the **O(b^m)** nodes in the search tree, where m is the maximum depth of any node.
- ❖ Note that m can be infinite if the tree is unbounded

Space Complexity: Needs linear space, O(bm)

- ❖ Depth-first tree search needs to store only a single path from the root to a leaf node, along with the remaining unexpanded sibling nodes for each node on the path.
- ❖ Once a node has been expanded, it can be removed from memory as soon as all its descendants have been fully explored. (See Figure 3.7.)
- ❖ For a state space with branching factor b and maximum depth m, depth-first search requires storage of only **O(bm)** nodes.
- ❖ DFS requires very less memory as it only needs to store a stack of the nodes on the path from root node to the current node.

3. Depth-Limited Search Algorithm

Failure of depth-first search can be alleviated by supplying depth-first search with a predetermined depth limit ℓ. That is, nodes at depth ℓ are treated as if they have no successors. This approach is called depth-limited search. The depth limit solves the infinite-path problem.

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

a. **Standard failure value**: It indicates that the problem have no solution.
b. **Cutoff failure value:** It indicates no solution within a given depth limit.

Example: In the figure below, the depth-limit is 1. So, only level 0 and 1 get expanded in **A->B->C .** DFS sequence, starting from the root node A till node C. It is not giving satisfactory results because we could not reach goal node I.

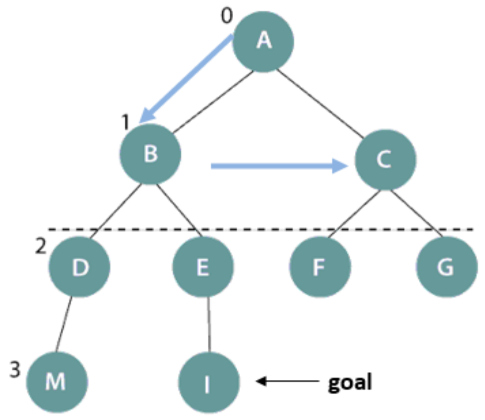


Figure 3.8: Search strategies of depth limited search algorithm

Evaluation of Depth Limited Search Algorithm

- **Completeness**: Incomplete if we choose $\ell < d$, that is, the shallowest goal is beyond the depth limit.
- **Optimal:** it is a special case of the DFS and nonoptimal if we choose $\ell > d$.
- **Time Complexity:** Time complexity of the DLS algorithm is $O(b^{\ell})$. Where ℓ is depth limit.
- **Space Complexity:** Space complexity of the DLS algorithm is $O(b\ell)$.

4. Depth First Iterative Deepening Search (DFIDS)

DFIDS is used in combination with depth-first tree search, that finds the best depth limit. It does this by gradually increasing the limit—first 0, then 1, then 2, and so on—until a goal is found. Iterative deepening combines the benefits of depth-first and breadth-first search. Iterative deepening search may seem wasteful because states are generated multiple times until the goal node reaches. It is the preferred uninformed search method when the search space is large and the depth of the solution is not known.

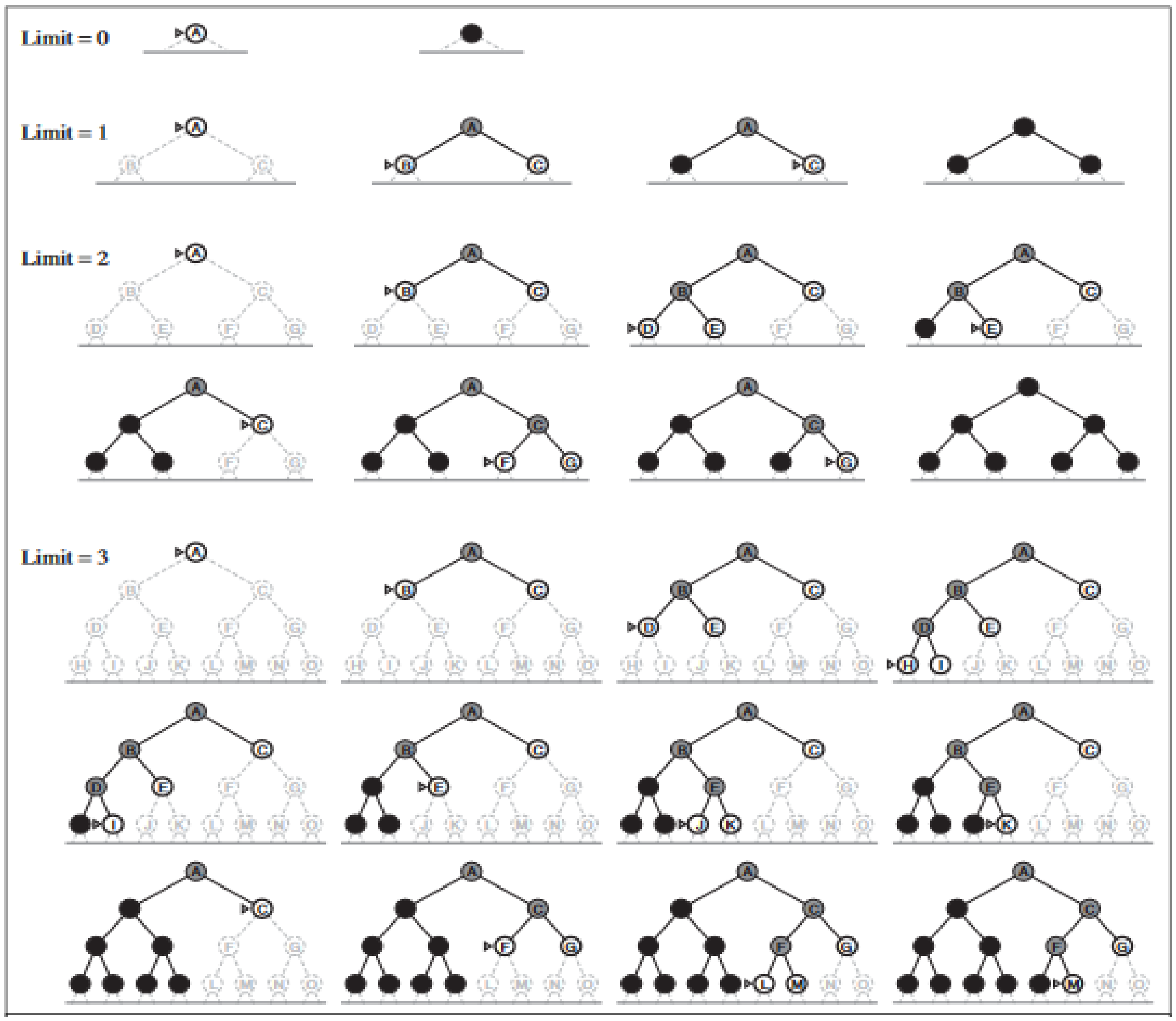


Figure 3.9: Four iterations of iterative deepening search on a binary tree.

- Combines advantages of both ***breadth-first and depth-first search***
- It continuously increments the depth limit by one until a solution is found.
- By using a depth-first approach on every iteration, iterative deepening search first avoids the ***memory cost of breadth-first search***.

Evaluation of Iterative Deepening Search

- Complete? Yes
- Time? $db + (d - 1)b^2 + : : : +(1)b^d = O(b^d)$
- Space? O(bd)
- Optimal? Yes, if step cost = 1
- The time complexity is O(bd) as in BFS, which is better than DFS

3.6.3 Informed Search Strategies

Informed search strategy uses problem-specific knowledge beyond the definition of the problem itself. It can find solutions more efficiently than an uninformed strategy. Also called heuristic search or Best-first search.

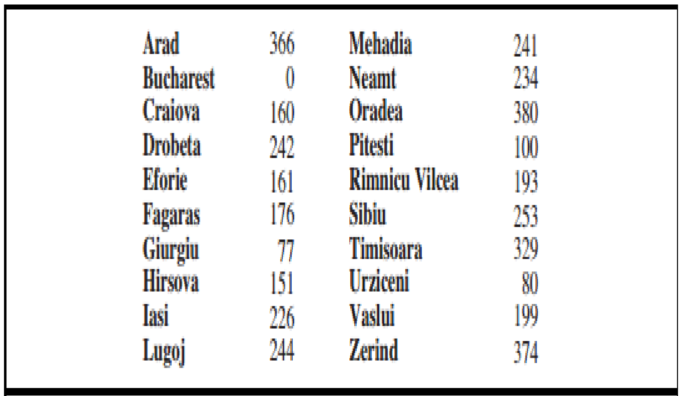
- Best-first search is an instance of the general TREE-SEARCH algorithm.
- A node is selected for expansion based on an evaluation function, f(n).
  - f(n) tells you the approximate distance of a node n, from a goal node
  - f(n) is constructed as a cost estimate, so the node with the lowest evaluation is expanded first.
- The choice of f determines the search strategy.
- Best-first algorithms include as a component of the **f**, **heuristic function**, denoted **h(n)**:
  - *h(n) = estimated cost of the cheapest path from the state at node n to a goal state.*
- A Heuristic is an operationally-effective bit of information on how to direct search in a problem space.
- n is a goal node, then h(n) = 0.

Types of Informed Search Algorithms

1. **Greedy best-first search**
   - It expands the node that is closest to the goal
   - evaluates nodes by using just the heuristic function;
     - i.e., f(n) = h(n).
     - “greedy”—at each step it tries to get as close to the goal as it can.

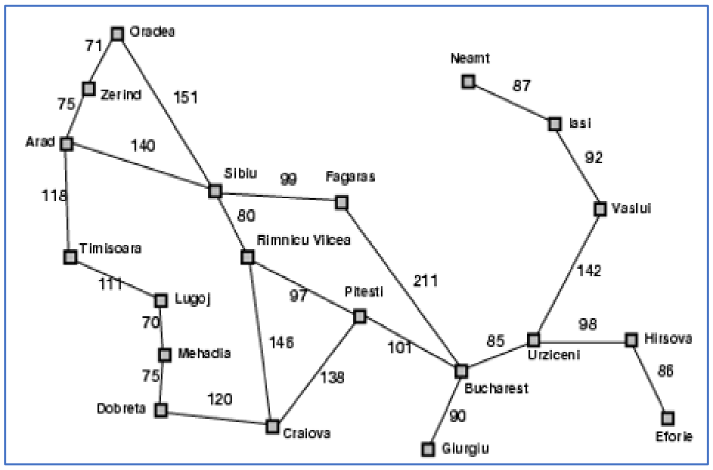
- **Problem:** Find the routes that gets from Arad to Bucharest.

- We use the straight-line distance as a heuristic: $h_{SLD}(n)$ = straight-line distance from city n to Bucharest.



Arad	366	Mehadia	241
Bucharest	0	Neamt	234
Craiova	160	Oradea	380
Drobeta	242	Pitesti	100
Eforie	161	Rimnicu Vilcea	193
Fagaras	176	Sibiu	253
Giurgiu	77	Timisoara	329
Hirsova	151	Urziceni	80
Iasi	226	Vaslui	199
Lugoj	244	Zerind	374

Figure 3.22 Values of h_{SLD}—straight-line distances to Bucharest.



Notice: The **heuristic value** of cannot be computed from the **problem description itself**.

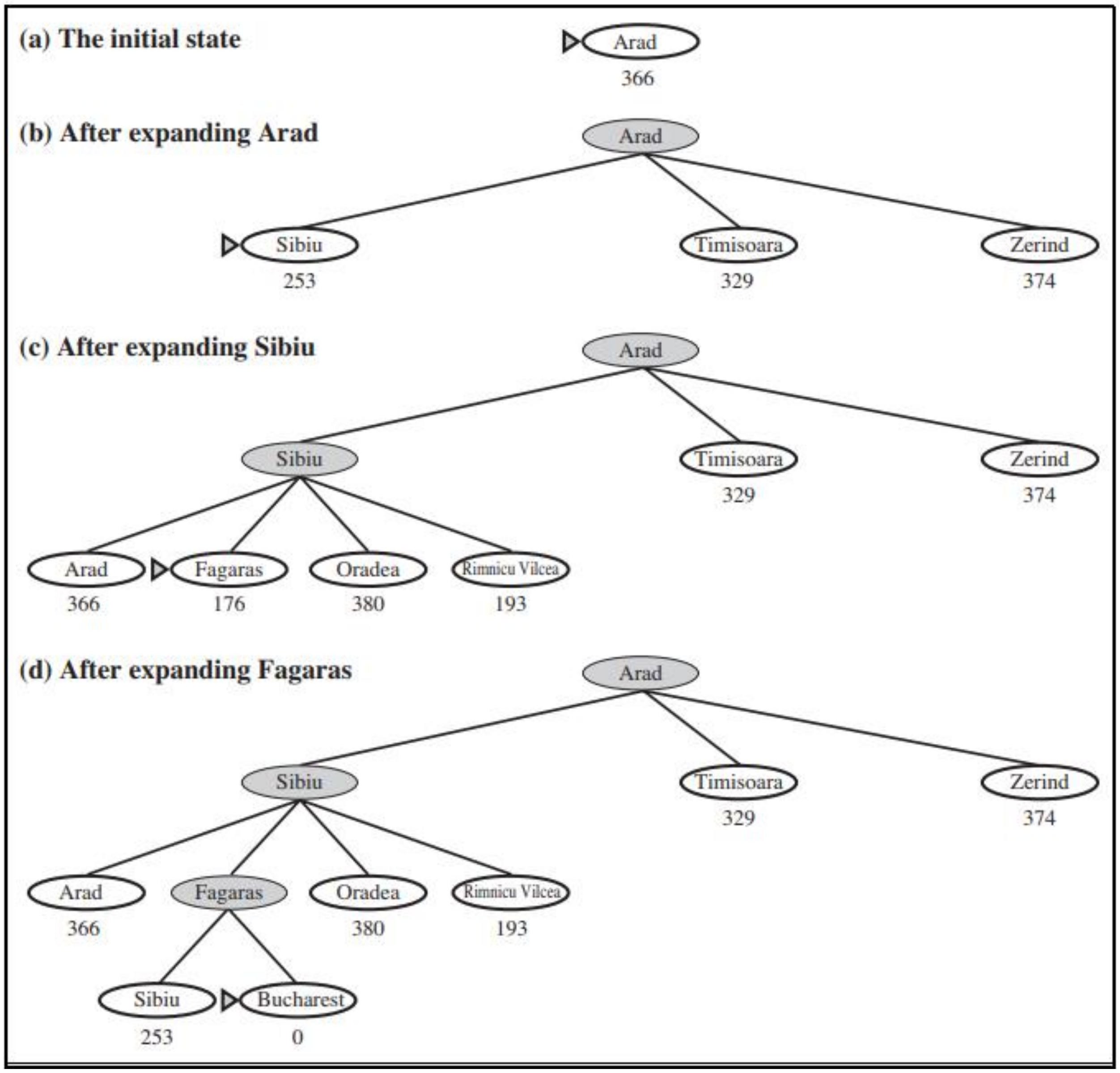


Figure 3.10 Stages in a greedy best-first tree search for Bucharest with the straight-line distance heuristic h_{SLD}. Nodes are labeled with their h-values.

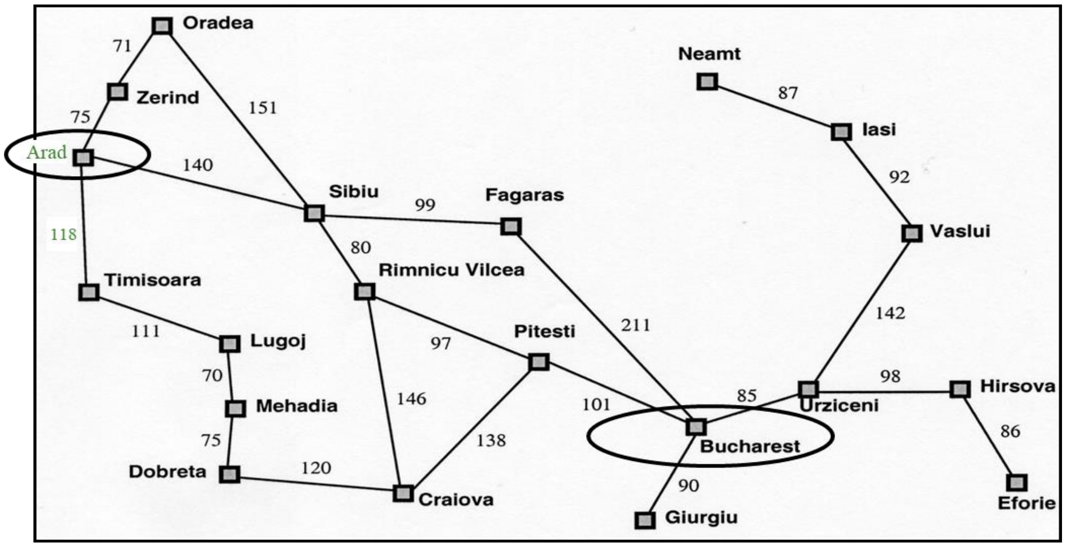
For this particular problem, greedy best-first search using h_{SLD}. finds a solution without ever expanding a node that is not on the solution path; hence, its search cost is minimal.

Evaluation of greedy best-first search

- **Optimal?**
  - Not optimal (not guaranteed to render lowest cost solution).
    - **Found**: Arad→ Sibiu→ Fagaras→ Bucharest (450km)

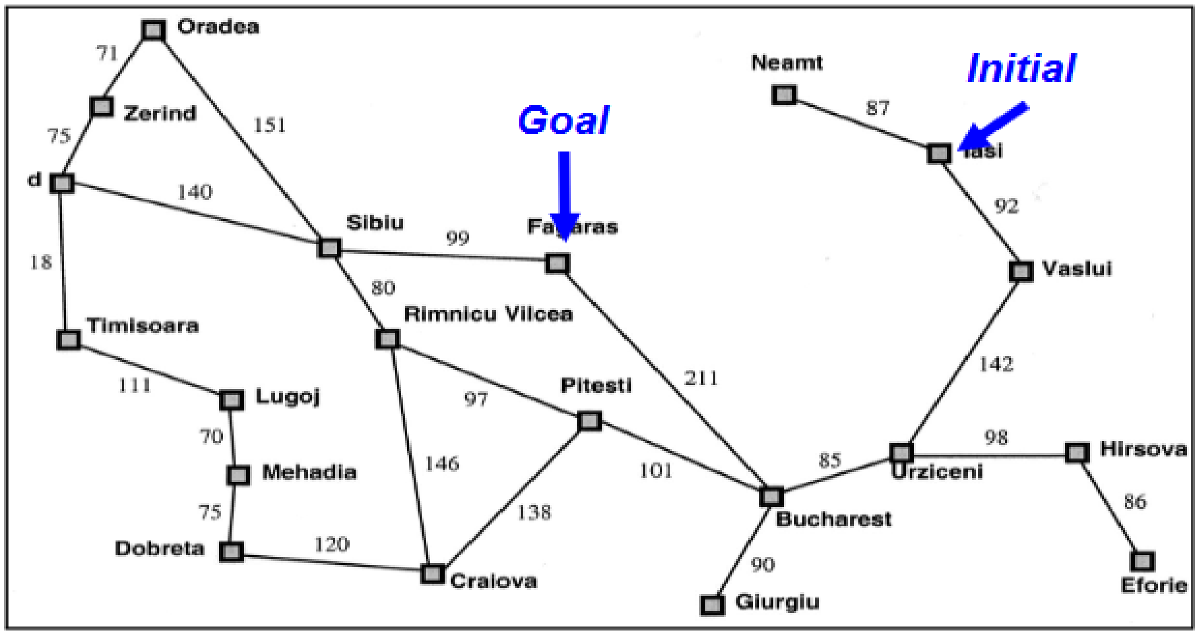
– **Shorter:** Arad→ Sibiu→ Rimnicu Vilcea→ Pitesti→ Bucharest (418km)

- This shows why the algorithm is called "**greedy**"—at each step it tries to get as close to the goal as it can.



► ***Complete?***

- No – can get stuck in loops,
- e.g., Iasi → Neamt → Iasi → Neamt → …



► ***Time?***

- ❑ *$O(b^m)$ – worst case* (like Depth First Search)
- ❑ But a good heuristic can give a dramatic improvement

► ***Space?***

- ❑ *$O(b^m)$* – keeps all nodes in memory

2. A* search Algorithm

A well-known form of best-first search and pronounced “**A-star search**”.

► **Key Idea:** avoid expanding paths that are already expensive, but expand the most promising first.

► It evaluates nodes by combining g(n), the cost to reach the node, and h(n), the cost to get from the node to the goal:

f(n) = g(n) + h(n)

► Expand the node n with the smallest f(n).

- ❖ g(n), the actual cost of the path from the start node to node n
- ❖ h(n), the estimated cost of the path to get from the node n to the goal node
- ❖ f(n), the estimated total cost of the path through n to goal.

► **Implementation**: **A* search algorithm** is implemented by using a **priority queue,** add the nodes in the frontier increasing order of cost f(n).

Example: Find the route that gets from Arad to Bucharest by using A* search algorithm.

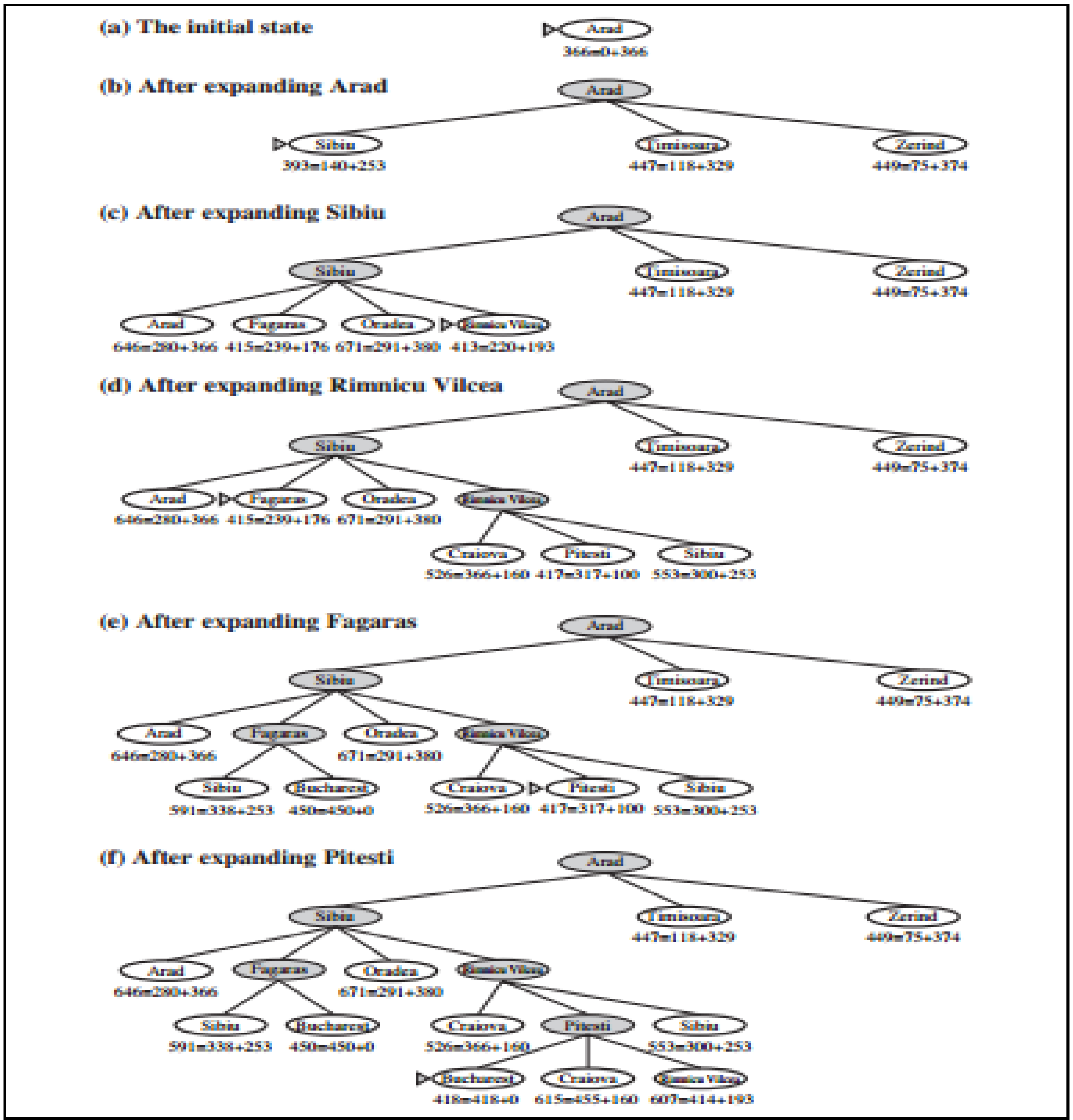


Figure 3.24 Stages in an A* search for Bucharest. Nodes are labeled with $f = g + h$. The h values are the straight-line distances to Bucharest taken from Figure 3.22.

Evaluation of A* search

- **Complete**: Yes (unless there are infinitely many nodes with. (if; f ≤ f(G)).
- **Optimality**: Yes (provided h admissible or consistent).
- **Time complexity**: Number of nodes expanded is still exponential in the length of the solution.
- **Space complexity**: Keeps all nodes in memory, so also exponential.

Activity 3.1

1. Use A* graph-search to generate a path from Lugoj to Bucharest using the straight-line distance heuristic.
2. Prove each of the following statements:
    a) Breadth-first search is a special case of uniform-cost search.
    b) Breadth-first search, depth-first search, and uniform-cost search are special cases of best-first search.
    c) Uniform-cost search is a special case of A* search.

3.7 Constraint satisfaction problem

Constraint satisfaction problem consists of different classes of problems that are solved under certain constraints or rules of the problem. Formally, Constraint satisfaction problem (or CSP) is defined by:

- ✓ set of variables, X1, X2, . . . , Xn, and
- ✓ set of constraints, C1, C2, . . . , Cm. which are followed by the set of variables
- ✓ Non-empty domain Di of possible values for each variable
    - D1, D2, ... Dn where Di = {v1, ..., vk}

- A state of the problem is defined by an assignment of values to all of the variables, {Xi = vi, Xj =vj , . . .**}.**
- **Consistent or legal assignment:** An assignment that does not violate any constraints.
- **Complete assignment**: is one in which every variable is assigned a value.
- **The Solution to a CSP:** is a complete assignment that satisfies all the constraints.

3.7.1 Types of Constraints

- **Unary constraint:** involves single variables
    - ❖ For example, SA actively dislikes the color green. **SA ≠ Green**
- **Binary constraint**: involves two variables.
    - ❖ Example, SA ≠ NSW is a binary constraint.
- **Higher-order constraints:** involve three or more variables.
    - ❖ Example: cryptarithmetic puzzles.

- **Preference constraints:** previous all constraints are absolute constraints, violation of which rules out a potential solution. Many real-world CSPs include preference constraints indicating which solutions are preferred.

Solving CSP: Map Coloring

Given a map of Australia showing its states and territories. We have been given the task of coloring each region either red, green, or blue in such a way that no neighboring regions have the same color.

Formulate this as CSP,

Variables: WA, NT , Q, NSW , V , SA, and T .

Domain of each variable is the set

Di ={red, green, blue}.

Constraints: Adjacent regions must have distinct colors.

• e.g., WA ≠ NT

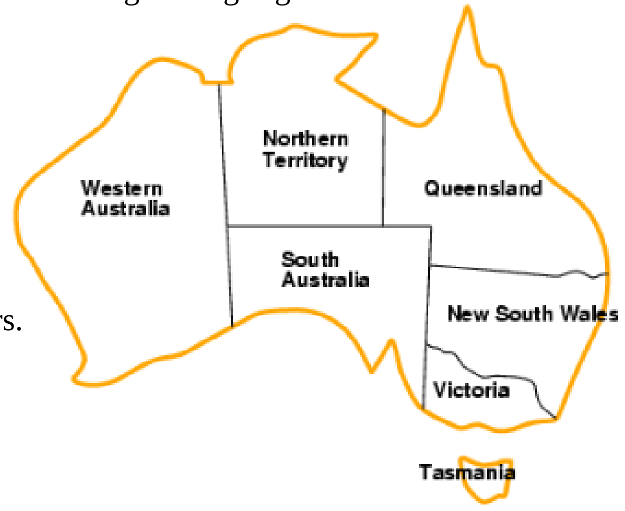
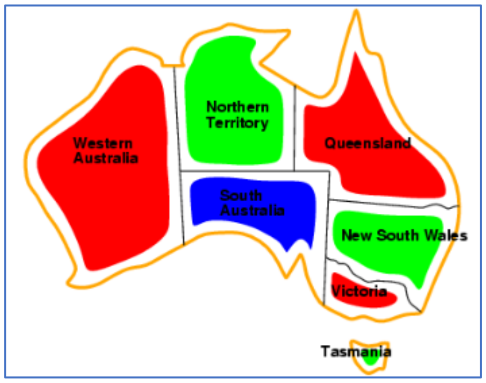


Figure 3.12: map of Australia

Solutions are **complete** (every variable have assigned a value) and **consistent** (satisfying all constraints) assignments. There are many possible solutions, such as

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



Real world CSP Problems

- Course offering problem
  - Who teaches what course
- Timetable problem
  - Which class is offered when and where?
- Transportation scheduling

A simple backtracking algorithm for constraint satisfaction problems

```
function BACKTRACKING-SEARCH(csp) returns a solution, or failure
  return RECURSIVE-BACKTRACKING({ }, csp)

function RECURSIVE-BACKTRACKING(assignment, csp) returns a solution, or failure
  if assignment is complete then return assignment
  var ← SELECT-UNASSIGNED-VARIABLE(VARIABLES[csp], assignment, csp)
  for each value in ORDER-DOMAIN-VALUES(var, assignment, csp) do
     if value is consistent with assignment according to CONSTRAINTS[csp] then
        add {var = value} to assignment
        result ← RECURSIVE-BACKTRACKING(assignment, csp)
        if result ≠ failure then return result
        remove {var = value} from assignment
  return failure
```

Figure 3.13: A simple backtracking algorithm for constraint satisfaction problems.

CHAPTER 4

Knowledge and Reasoning

4.1 Introduction

Artificial Intelligence is the study and development of systems that demonstrate intelligent behavior. Based on the definition of AI, we can define Knowledge Representation & Reasoning as follows. The study of ways to represent knowledge and reason with information in order to achieve intelligent behavior. Knowledge representation & reasoning is the part of AI that is concerned with thinking and how thinking contributes to intelligent behavior.

- ***Knowledge representation***, then, is the field of study concerned with *using formal symbols* to *represent a collection of propositions* believed by some agent.
- **Reasoning**, it is the ***formal manipulation*** of the symbols representing a collection of believed propositions ***to produce representations of new ones***.
    - We might start with the sentences "*John loves Mary*" and "*Mary is coming to the party*" and after a certain amount of manipulation produces the sentence, "*Someone John loves is coming to the party*."
    - We would call this form of reasoning **logical inference** because the final sentence represents a logical conclusion of the propositions represented by the initial ones.
- Reasoning is a form of calculation, not unlike arithmetic, but over symbols standing for propositions rather than numbers.

4.2 Knowledge-Based Agents

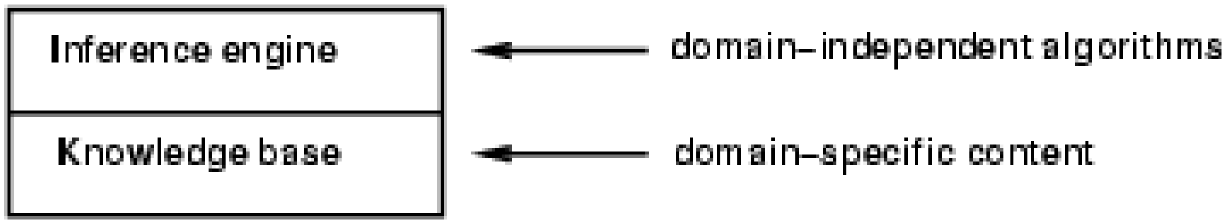
Knowledge-based agents are those agents who have the capability of maintaining an internal state of **knowledge**, reason over that **knowledge**, update their **knowledge** after observations and take actions. The central component of a Knowledge-Based Agent is a ***Knowledge-Base.*** A set of sentences in formal language. These sentences are expressed using a knowledge representation language.

There are two generic knowledge-based agent functions:

- TELL - add new sentences (facts) to the KB
    - "Tell it what it needs to know"
- ASK - query what is known from the KB

- "Ask what to do next"

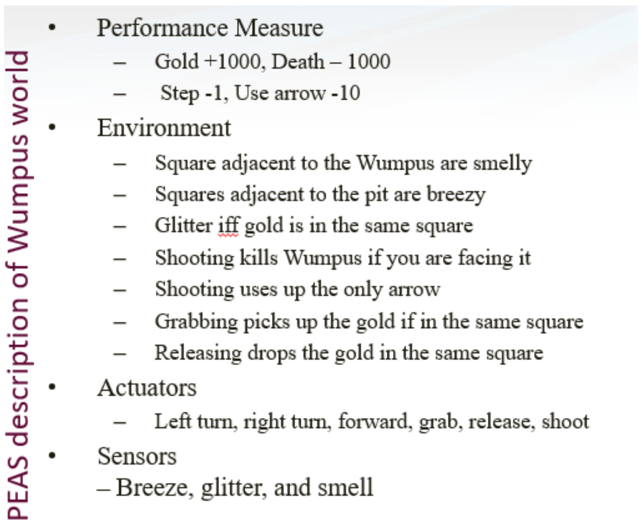
The Knowledge-based agent is composed of:



Approaches/methods of designing knowledge-based agents:

- Declarative
  - You can build a knowledge-based agent simply by "TELLing" it what it needs to know
- Procedural
  - Encode desired behaviours directly as program code

The demonstrative example of a knowledge-based agent: WUMPUS World



PEAS description of Wumpus world

- Performance Measure
  - Gold +1000, Death – 1000
  - Step -1, Use arrow -10
- Environment
  - Square adjacent to the Wumpus are smelly
  - Squares adjacent to the pit are breezy
  - Glitter iff gold is in the same square
  - Shooting kills Wumpus if you are facing it
  - Shooting uses up the only arrow
  - Grabbing picks up the gold if in the same square
  - Releasing drops the gold in the same square
- Actuators
  - Left turn, right turn, forward, grab, release, shoot
- Sensors
  – Breeze, glitter, and smell

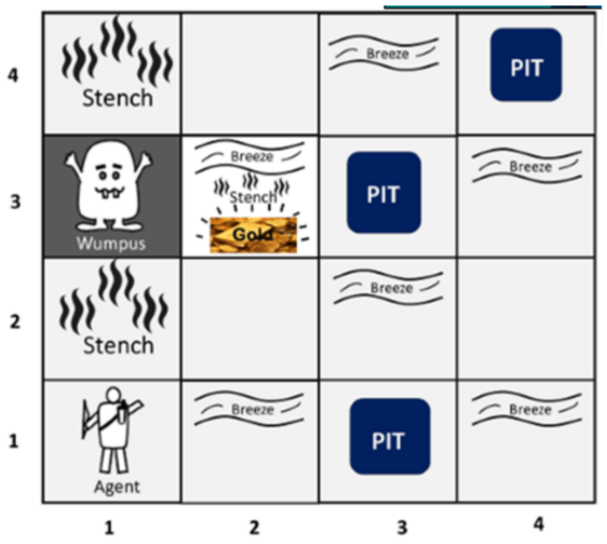


Figure 4.1: Wumpus word

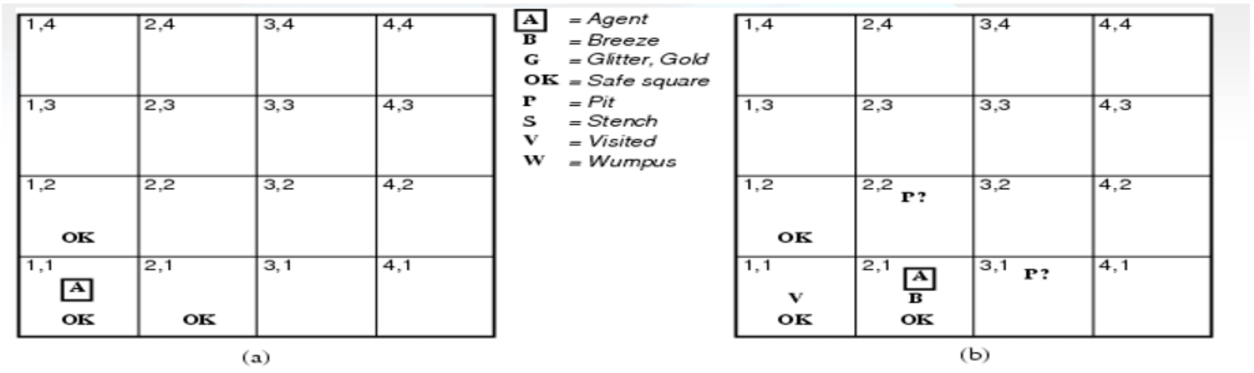
Wumpus World PEAS description

- ❖ Fully Observable? No – only local perception
- ❖ Deterministic? Yes – outcomes exactly specified
- ❖ Episodic? No – sequential at the level of actions
- ❖ Static? Yes – Wumpus and Pits do not move
- ❖ Discrete? Yes
- ❖ Single-agent? Yes – Wumpus is essentially a natural feature

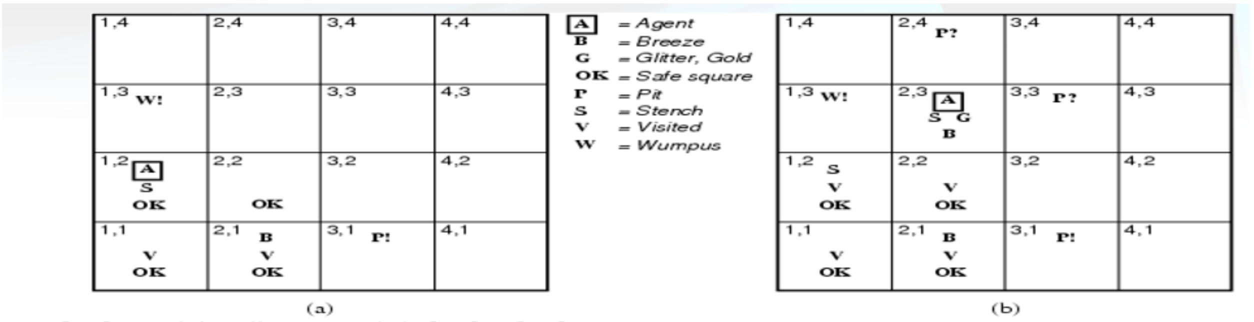
Exploring the Wumpus World

Now we will explore the Wumpus world and will determine how the agent will find its goal by applying logical reasoning.

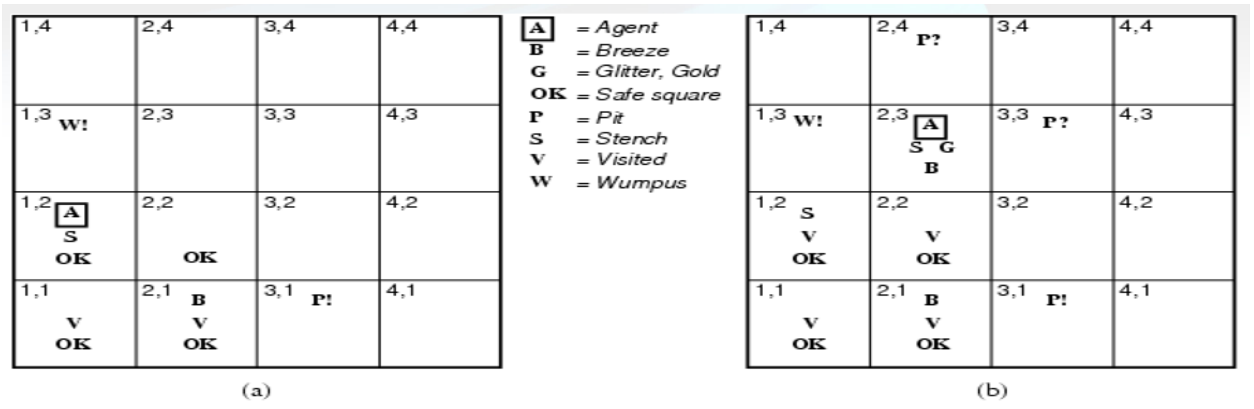
Note that in each case for which the agent draws a conclusion from the available information, that conclusion is guaranteed to be correct if the available information is correct. This is a fundamental property of logical reasoning.



1. The KB initially contains the rules of the environment.
2. [1,1] The first percept is [*none, none,none,none,none*], Move to safe cell e.g. 2,1
3. [2,1] Breeze indicates that there is a pit in [2,2] or [3,1] Return to [1,1] to try next safe cell



4. [1,2] Stench in cell: wumpus is in [1,3] or [2,2]
   YET … not in [1,1]
      Thus … not in [2,2] or stench would have been detected in [2,1]
      Thus … wumpus is in [1,3]
      Thus … [2,2] is safe because of lack of breeze in [1,2]
      Thus … pit in [3,1] and Move to next safe cell [2,2]



5. [2,2] Detect nothing
   Move to unvisited safe cell e.g. [2,3]
6. [2,3] Detect glitter , smell, breeze
   Thus… pick up gold
   Thus… pit in [3,3] or [2,4]

How can knowledge be represented?

- There are mainly four ways of knowledge representation
  - Logical Representation
  - Semantic Representation
  - Frame Representation
  - Production Rules

4.3 Logical Representation and Reasoning

This section summarizes the fundamental concepts of logical representation and reasoning. Logics are formal languages for representing information such that conclusions can be drawn. Logic is widely used as a representational method for Artificial Intelligence. It can be defined as the ***proof or validation*** behind any ***reason*** provided. It is a language with some concrete rules which deal with **propositions** & **has no ambiguity** in representation.

- It consists of precisely defines ***Syntax*** and ***Semantics***.
- Each sentence can be translated into logic using syntax and semantics.
  - ***Syntax***: defines well- formed sentences in the language
  - ***Semantics***: defines the truth or meaning of the sentence in the world
- Logic representation can be explained as:
  - Propositional logic
  - Predicate logic

4.3.1 Propositional Logic (PL)

It Is the simplest logic, declarative statement, either it can be true or false. It is a technique of knowledge representation in logical and mathematical form.

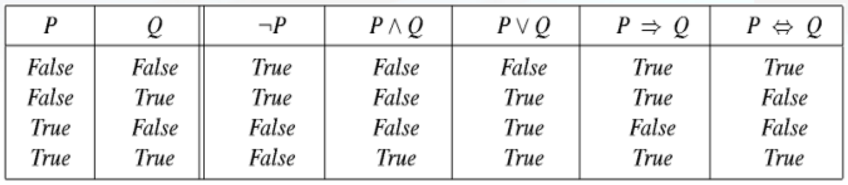
- **Syntax**
  - Propositions, e.g. "it is wet"
  - Logical Connectives: and (conjunction), or(disjunction), not (negation), implies (implication), iff (biconditional)
  - Brackets, T (true) and F (false)
- **Semantics**
  - Define how connectives affect the truth
  - "P and Q" is true if and only if P is true and Q is true
  - Use **truth tables** to work out the truth of statements

Logical connectives

Logical connectives are used to connect two simpler propositions or representing a sentence logically.

1. **Negation:** A sentence such as ¬ P is called the negation of P. A literal can be either Positive literal or negative literal.
2. **Conjunction:** A sentence that has ∧ connective such as, **P ∧ Q** is called a conjunction. **Example:** Rohan is intelligent and hardworking. It can be written as, P= Rohan is intelligent, Q= Rohan is hardworking. → P∧ Q.
3. **Disjunction:** A sentence which has ∨ connective, such as **P ∨ 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 Engineer, so we can write it as **P ∨ Q**.
4. **Implication:** A sentence such as P → 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 → Q
5. **Biconditional:** A sentence such as P⇔ 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 ⇔ Q.

- Truth Table



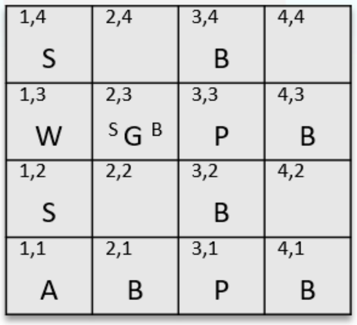
P	Q	$\neg P$	$P \wedge Q$	$P \vee Q$	$P \Rightarrow Q$	$P \Leftrightarrow Q$
False	*False*	*True*	*False*	*False*	*True*	*True*
False	*True*	*True*	*False*	*True*	*True*	*False*
True	*False*	*False*	*False*	*True*	*False*	*False*
True	*True*	*False*	*True*	*True*	*True*	*True*

- E.g.
  P – It is hot
  Q – It is humid
  R – It is raining

- Conditions
  ✓ If it is humid, then it is hot Q→P
  ✓ If it is hot and humid then it is not raining (P Λ Q) → ¬R

Example: *WUMPUS worlds using propositional logic*

$P_{x,y}$ is true if there is a pit in $[x, y]$.
$W_{x,y}$ is true if there is a wumpus in $[x, y]$, dead or alive.
$B_{x,y}$ is true if the agent perceives a breeze in $[x, y]$.
$S_{x,y}$ is true if the agent perceives a stench in $[x, y]$.



There is no pit in [1,1]:
R1: $\neg P_{1,1}$
A square is breezy if and only if there is a pit in a neighbouring square.
R2: $B_{1,1} \Leftrightarrow (P_{1,2} \vee P_{2,1})$.
R3: $B_{2,1} \Leftrightarrow (P_{1,1} \vee P_{2,2} \vee P_{3,1})$
The preceding sentences are true in all Wumpus worlds
R4 : $\neg B_{1,1}$
R5 : $B_{2,1}$

Limitations of Propositional Logic

- We cannot represent relations like ALL, some, or none with propositional logic. Example:
  - **All the girls are intelligent.**
  - **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.

4.3.2 First Order Predicate Logic

FOL is another way of knowledge representation in AI. It is an extension to PL (Propositional Logic). It is a powerful language that develops information about the object more easily and can also express the relationship between those objects. FOL doesn't only assume that the world contains facts like PL but also assume

– **Objects**: A, B, people, numbers, colors, wars, theories, pits, wumpus, ...

- **Relations**: It can be unary relation such as: red, round, is adjacent, or n-any relation such as: the sister of, brother of, has color, comes between
- **Function**: Father of, best friend, third inning of, end of, ...

- As a natural language, first-order logic also has two main parts:
  - **Syntax**
  - **Semantics**
- The syntax of FOL determines which collection of symbols is a logical expression in first-order logic. The basic syntactic elements of FOL are symbols.
- More expressive logic than propositional
  - **Constants** are objects: john, apples
  - **Predicates** are properties and relations:
    - likes(john, apples)
  - **Functions** transform objects:
    - likes(john, fruit_of(apple_tree))
  - **Variables** represent any object: likes(X, apples)
  - **Quantifiers** qualify values of variables
    - True for all objects (Universal): $\forall X.$ likes(X, apples)
    - Exists at least one object (Existential): $\exists X.$ likes(X, apples)

4.3.2.1 Quantifiers in First-order logic

A quantifier is a language element that **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 quantifiers.

- Universal Quantifier, (for all, everyone, everything)
- Existential quantifier, (for some, at least one).

a. 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. It is represented by a symbol ∀, which resembles an inverted A. In universal quantifier we use implication " ⇒ ". If x is a variable, then ∀x is read as:

- For all x

- For each x
- For every x.

Example: All men drink coffee.

- Let a variable x which refers to a man so all x can be represented in**: ∀x man(x) → drink (x, coffee).**
- It will be read as: *There are all x where x is a man who drink coffee*.

b. Existential quantifier: which express that the statement within its scope is true for at least one instance of something. It is denoted by the logical operator ∃, which resembles as inverted E. In the Existential quantifier we always use AND or Conjunction symbol (∧). If x is a variable, then existential quantifier will be ∃x or ∃(x). And it will be read as:

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

Example: Some boys are intelligent.

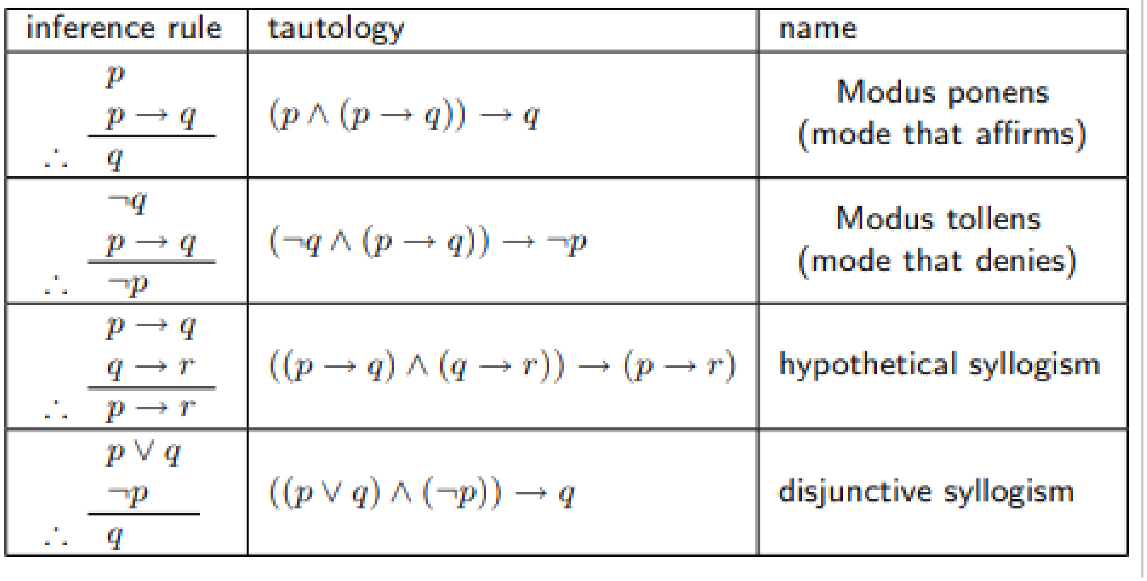
- ∃x: boys(x) ∧ intelligent(x)
- It will be read as: There are some x where x is a boy who is intelligent.

4.4 Inference

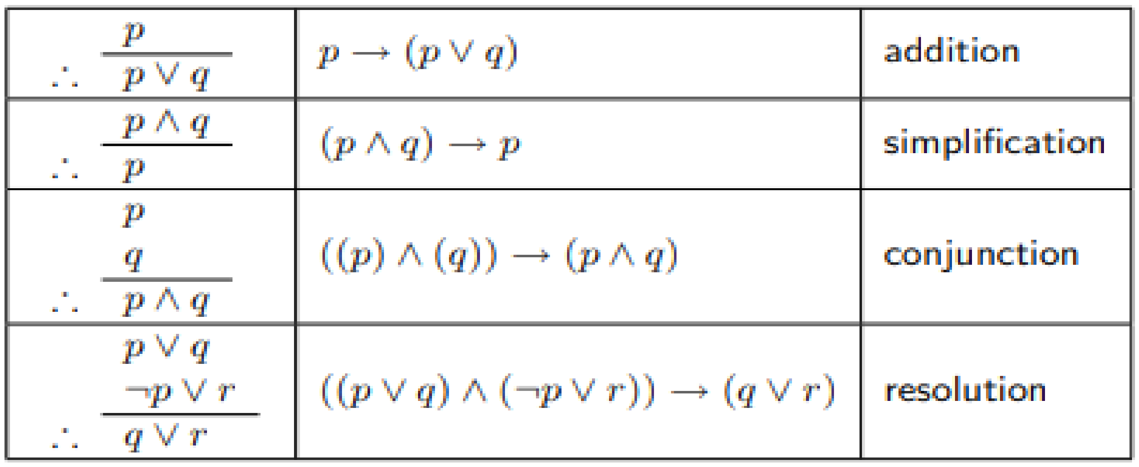
In artificial intelligence, we need intelligent computers which can create new logic from existing ones or by evidence, **so generating the conclusion from evidence and facts is termed as Inference.**

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

4.4.1 Inference in Propositional Logic



inference rule	tautology	name
$\begin{array}{l} p \\ p \to q \\ \hline \therefore\ q \end{array}$	$(p \land (p \to q)) \to q$	Modus ponens (mode that affirms)
$\begin{array}{l} \neg q \\ p \to q \\ \hline \therefore\ \neg p \end{array}$	$(\neg q \land (p \to q)) \to \neg p$	Modus tollens (mode that denies)
$\begin{array}{l} p \to q \\ q \to r \\ \hline \therefore\ p \to r \end{array}$	$((p \to q) \land (q \to r)) \to (p \to r)$	hypothetical syllogism
$\begin{array}{l} p \lor q \\ \neg p \\ \hline \therefore\ q \end{array}$	$((p \lor q) \land (\neg p)) \to q$	disjunctive syllogism



$\begin{array}{l} p \\ \hline \therefore\ p \lor q \end{array}$	$p \to (p \lor q)$	addition
$\begin{array}{l} p \land q \\ \hline \therefore\ p \end{array}$	$(p \land q) \to p$	simplification
$\begin{array}{l} p \\ q \\ \hline \therefore\ p \land q \end{array}$	$((p) \land (q)) \to (p \land q)$	conjunction
$\begin{array}{l} p \lor q \\ \neg p \lor r \\ \hline \therefore\ q \lor r \end{array}$	$((p \lor q) \land (\neg p \lor r)) \to (q \lor r)$	resolution

Activity 4.1

Which rule of inference is used in each argument below?

1. Alice is a Math major. Therefore, Alice is either a Math major or a CSI major.
2. Jerry is a Math major and a CSI major. Therefore, Jerry is a Math major.
3. If it is rainy, then the pool will be closed. It is rainy. Therefore, the pool is closed.
4. If it snows today, the university will close. The university is not closed today. Therefore, it did not snow today.
5. If I go swimming, then I will stay in the sun too long. If I stay in the sun too long, then I will sunburn. Therefore, if I go swimming, then I will sunburn.
6. l go swimming or eat an ice cream. I did not go swimming. Therefore, I eat an ice cream

4.4.2 Inference in FOL

Inference in FOL is used to deduce new facts or sentences from existing sentences

FOL inference rules for quantifier:

Basic inference rules in FOL are:

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

1. Universal Generalization:

- is a valid inference rule which states that if premise P(c) is true for any arbitrary element c in the universe of discourse, then we can have a conclusion as ∀ x P(x).
- It can be represented as: $\frac{P(c)}{\forall x \, P(x)}$
- This rule can be used if we want to show that every element has a similar property.
- Example:
  - Let's represent, P(c): "A byte contains 8 bits", so for ∀ x P(x) "All bytes contain 8 bits.", it will also be true.

2. Universal Instantiation:

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

3. Existential Instantiation:

- is also called as Existential Elimination, which is a valid inference rule in FOL.
- It can be applied only once to replace the existential sentence.
- The new KB is not logically equivalent to old KB, but it will be satisfiable if old KB was satisfiable.
- This rule states that one can infer P(c) from the formula given in the form of ∃x P(x) for a new constant symbol c.
- The restriction with this rule is that c used in the rule must be a **new term** for which P(c) is true.
- It can be represented as: $\frac{\exists x\, P(x)}{P(c)}$
- Example:
    - From the given sentence: ∃x Crown(x) ∧ OnHead(x, John),
        - So we can infer: Crown(K) ∧ OnHead(K, John), as long as K does not appear in the knowledge base.
        - The above used K is a constant symbol, which is called Skolem constant.

5. Existential introduction

- An existential introduction is also known as an existential generalization, which is a valid inference rule in first-order logic.
- This rule states that if there is some element c in the universe of discourse which has a property P, then we can infer that there exists something in the universe which has the property P.
- It can be represented as: $\frac{P(c)}{\exists x P(x)}$
- Example: Let's say that, “John got good marks in English."
    - "Therefore, someone got good marks in English."

4.5 Designing Expert System

The **expert system** is an intelligent computer program that solves the problems in a **specialized domain** that normally requires **human expertise**.

- ❖ Expert system manipulates the encoded kgs to solve the problems, make decisions or draw the conclusion, and answer the queries.
- ❖ An expert system’s knowledge is obtained from expert sources such as: -

- Textbooks, journal articles, databases, domain experts etc.

- Once a sufficient body of expert knowledge has been acquired,
  - it must be encoded in a form suitable for the system to use in its inference or reasoning processes.
  - It must be loaded into the knowledge base, then tested, and refined.

4.5.1 Components of an expert system

The components of the expert system consist of four major parts.

1. User Interface

- It enables the users to enter instructions and information into the expert system and to receive information from it.

2. Knowledge Base

- Knowledge base contains a really high-quality and extraordinary knowledge in a particular domain. In short, kg base is the database of facts and rules.
- An exceptionally remarkable knowledge is required to model intelligent systems.
- Kg base must be accurate and also the bug-free b/c the whole success of an expert system depends upon the knowledge.

3. Inference engine

- The **inference engine** of the expert system is the rule that defines how the expert process interprets the knowledge in an appropriate manner.
- In simple terms, the inference engine takes the knowledge base and then it applies forward chaining or backward chaining and it comes out with a conclusion.

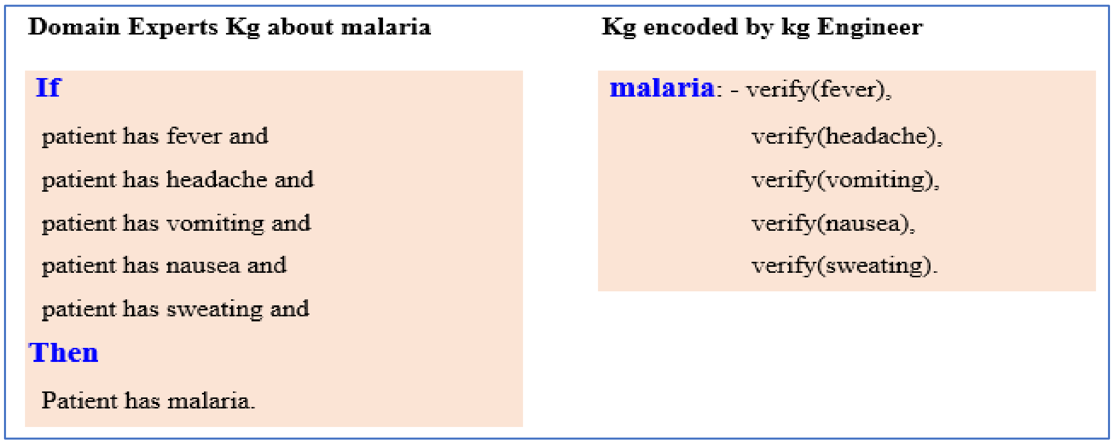
4. Development Engine

- Development engine is used to create the expert system. This process usually involved building the rule set.
- There are two basic approaches-
  - **Programming Language:** An ES can be created using programming language, e.g., Lisp, Prolog.
  - **Expert system shell:** a readymade processor that can be tailored to a specific problem domain through the addition of the appropriate Knowledge base.

4.5.3 Mini Project: Medical Diagnosis Expert System

The rationale of designing medical diagnosis expert system

Getting the right diagnosis is a key aspect of healthcare. Diseases should be treated well and at the right time. Diagnostic errors may lead to negative health outcomes, psychological distress, and financial costs. If a diagnostic error occurs, inappropriate or unnecessary treatment may be given to a patient, and these problems may become the cause of losing precious life. In an effort to address such a problem, this project/study attempted to design and develop expert systems which can provide advice and support for physicians and patients to facilitate the diagnosis process and recommend treatment of patients.



Domain Experts Kg about malaria	Kg encoded by kg Engineer
If patient has fever and patient has headache and patient has vomiting and patient has nausea and patient has sweating and **Then** Patient has malaria.	**malaria**: - verify(fever), verify(headache), verify(vomiting), verify(nausea), verify(sweating).

Sample codes of the Medical Diagnosis Expert System

```
/* Medical Diagnosis Expert System */
go:-hyphothesis(Disease),
   write("The patient have desease:"),
   write(Disease), nl,
   undo.
/* hyphothesis that have been tested */
hyphothesis(malaria):-malaria, !.
hyphothesis(cold):-cold, !.
hyphothesis(measles):-measles, !.
hyphothesis(covid):-covid,!.
hypothesize(unknown).             /* no diagnosis */
/*  hyphothesis disease identification rule */
malaria:-verify(fever),
       verify(headache),
       verify(vomiting),
       verify(nausen),
       verify(sweating).
cold:-verify(runny_nose),
       verify(sneezing),
       verify(headache),
       verify(sore_throat),
measles:-verify(runny_nose),
       verify(sneezing),
       verify(conjunctivitis),
       verify(rash),
       verify(cough).
covid:- verify(sore_throat),
       verify(tiredness),
       verify(shortness_of_breath),
       verify(dry_cough).
ask(Question):-write("Does the patient have: "),
                write(Question), nl,
                read(Response),nl,
                ((Response=yes; Response==y)
                ->assert(yes(Question));
                   assert(no(Question)), fail).
/* A dynamic predicate is introduced using dynamic/1, after which clauses may be added using
assertz/1  */
   :-dynamic yes/1,no/1.
/* verify the Symptom of Disease from the patient */
verify(Symptom):-(yes(Symptom)-> true; (no(Symptom)->fail; ask(Symptom))).
/* undo all yes and no assertions */
undo:-retract(yes(_)), fail.
undo:- retract(no(_)), fail.
undo.
```

CHAPTER 5

Learning

5.1 Overview of learning agents

An agent is learning, it improves its performance on future tasks after making observations about the world. Three main reasons why would we want an agent to learn:

- The designers cannot anticipate all possible situations that the agent might find itself in.
- The designers cannot anticipate all changes over time
- Sometimes human programmers have no idea how to program a solution themselves.

Any component of an agent can be improved by learning from data. The improvements, and the techniques used to make them, depend on four major factors:

- Which component is to be improved?
- What prior knowledge the agent already has?
- What representation is used for the data and the component?
- What feedback is available to learn from?

Example: An agent training to become a taxi driver.

- ✓ Every time the instructor shouts "Brake!" the agent might learn a condition– action rule for when to brake (component 1); the agent also learns every time the instructor does not shout.
- ✓ By seeing many camera images that it is told contain buses, it can learn to recognize them (2).
- ✓ By trying actions and observing the results—for example, braking hard on a wet road—it can learn the effects of its actions (3).
- ✓ Then, when it receives no tip from passengers who have been thoroughly shaken up during the trip, it can learn a useful component of its overall utility function (4).

5.2 Forms of learnings

Supervised Learning
Unsupervised Learning
Reinforcement Learning

5.2.1 Supervised Learning

- The machine learns a function from a set of training examples which are pre-classified feature vectors.
- Supervised learning involves the training of the model on a labeled dataset.
- Supervised machine learning includes two major processes: classification and regression.
  - **Classification** is the process where incoming data is labeled based on past data samples and manually trains the algorithm to recognize certain types of objects and categorize them accordingly.
    - **Classification techniques** predict discrete responses—for example, whether an email is genuine or spam, or whether a tumor is cancerous or benign. Classification models classify input data into categories
  - **Regression** is the process of identifying patterns and calculating the predictions of continuous outcomes. The system has to understand the numbers, their values, grouping (for example, heights and widths), etc. It is used to estimate real values (cost of houses, number of calls, total sales etc.) based on a continuous variable(s).

Use cases

- Image classification and segmentation
- Disease identification and medical diagnosis
- Fraud detection
- Spam detection

- Speech recognition
- Hand-written recognition uses classification to recognize letters and numbers

Examples of supervised learning

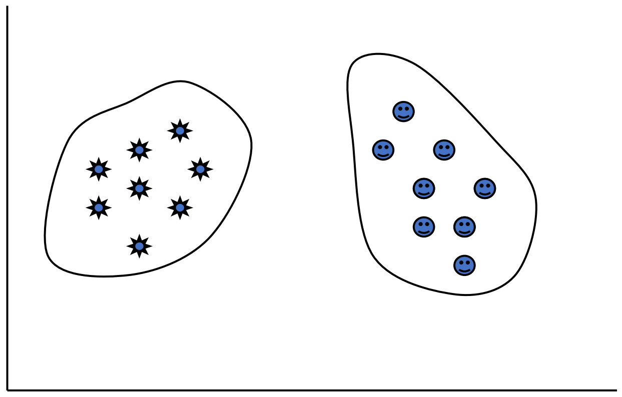
feature vector	**class**
(shape, color)	
(square, red)	I
(square, blue)	I
(circle, red)	II
(circle blue)	II
(triangle, red)	I
(triangle, green)	I
(ellipse, blue)	II
(ellipse, red)	II

(circle, green) ?
(triangle, blue)?

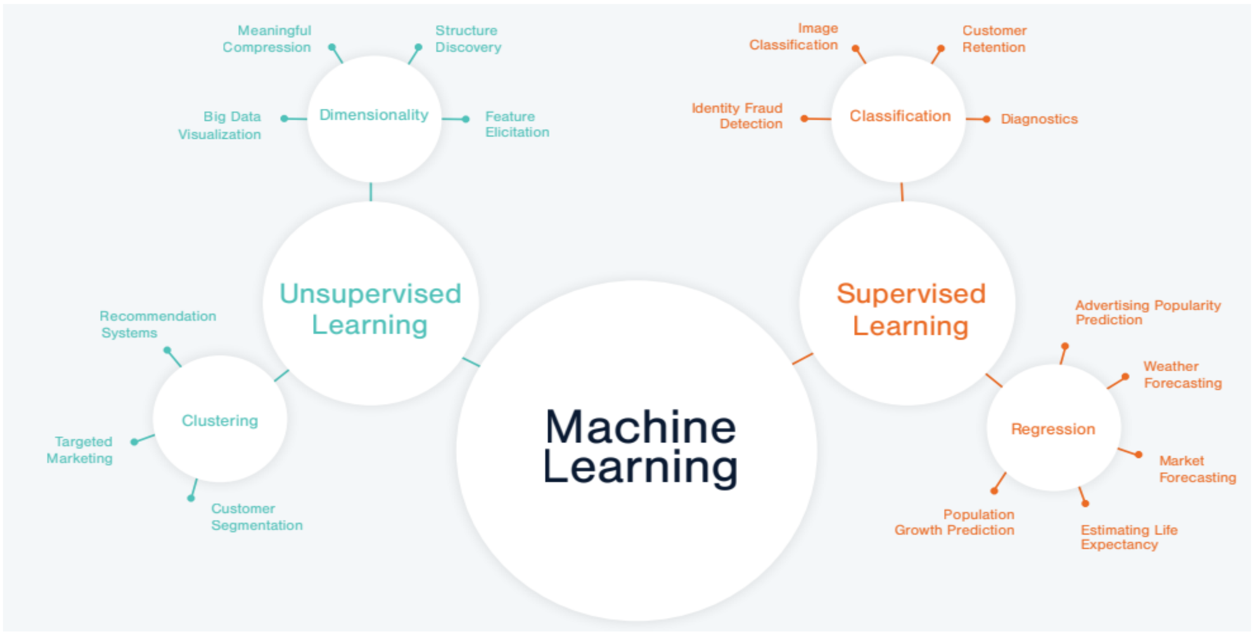
Given a previously unseen feature vector, what is the rule that tells us if it is in class I or class II?

5.2.2 Unsupervised Learning

- No classes are given. The idea is to find patterns in the data. This generally involves clustering.
- Unsupervised learning involves the training of a model in an unlabeled dataset. The model learns on its own by learning the features of the training dataset. Based on that learning features, the model makes predictions on test data.



- Unsupervised learning algorithms apply the following techniques to describe the data:
    1. **Clustering**: it is an exploration of data used to segment it into meaningful groups (i.e., clusters) based on their internal patterns without prior knowledge of group credentials. The credentials are defined by the similarity of individual data objects and also aspects of their dissimilarity from the rest (which can also be used to detect anomalies).
    2. **Dimensionality reduction**: there is a lot of noise in the incoming data. Machine learning algorithms use dimensionality reduction to remove this noise while distilling the relevant information.



5.2.3 Reinforcement Learning

- Machines learn from feedback after a decision is made.
- the agent must learn from reinforcement (reward or punishment)

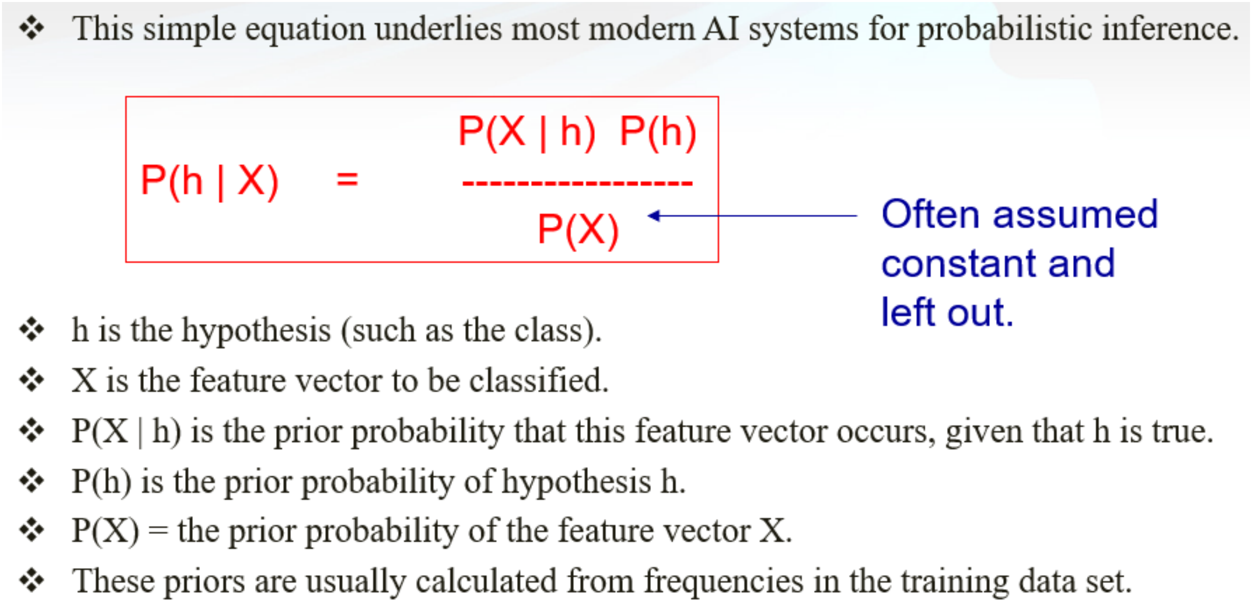
5.3 Learning Probabilistic Models

Agents can handle **uncertainty** by using the methods of **probability** and **decision theory**, but first they must learn their probabilistic theories of the world from **experience.** Probabilistic models are statistical models.

Bayesian Learning

- ➢ Bayesian learning: simply calculates the probability of the hypothesis and it makes predictions on that basis.
- ➢ That is the predictions are made by using all the hypotheses.
- ➢ The probability of each hypothesis is obtained by *Bayes' rule*.

Bayes' Rule



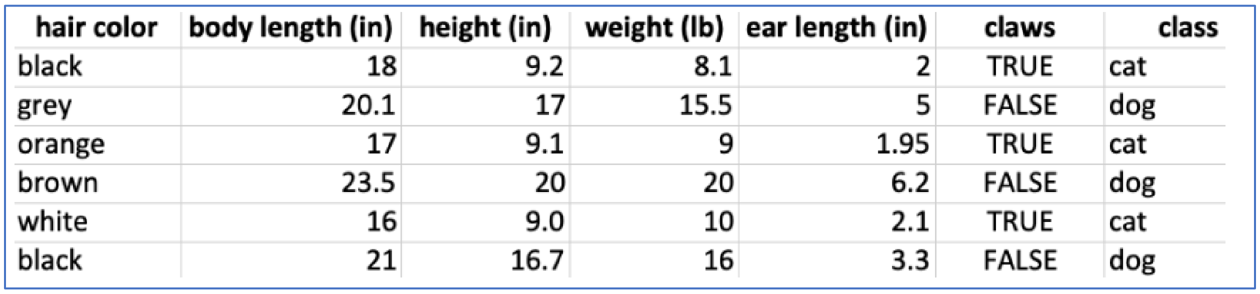
- ❖ This simple equation underlies most modern AI systems for probabilistic inference.

$$P(h \mid X) = \frac{P(X \mid h)\ P(h)}{P(X)}$$

Often assumed constant and left out.

- ❖ h is the hypothesis (such as the class).
- ❖ X is the feature vector to be classified.
- ❖ P(X | h) is the prior probability that this feature vector occurs, given that h is true.
- ❖ P(h) is the prior probability of hypothesis h.
- ❖ P(X) = the prior probability of the feature vector X.
- ❖ These priors are usually calculated from frequencies in the training data set.

Example: Say that you have this (tiny) dataset that classifies animals into two classes: cat and dog.



hair color	body length (in)	height (in)	weight (lb)	ear length (in)	claws	class
black	18	9.2	8.1	2	TRUE	cat
grey	20.1	17	15.5	5	FALSE	dog
orange	17	9.1	9	1.95	TRUE	cat
brown	23.5	20	20	6.2	FALSE	dog
white	16	9.0	10	2.1	TRUE	cat
black	21	16.7	16	3.3	FALSE	dog

- ❖ probability of the example being a cat, given that hair color is black, body length is 18 inches, height is 9.2, weight is 8.1 lb, …
- ❖ The conditional probability is, generically, P(class | feature set). In our example, classes = {cat, dog} and feature set = {hair color, body length, height, weight, ear length, claws}.

Choosing Hypothesis

$$P(h|D) = \frac{P(D|h)P(h)}{P(D)}$$

Generally want the most probable hypothesis given the training data
Maximum a posteriori hypothesis h_{MAP}:

$$\begin{aligned} h_{MAP} &= \arg\max_{h \in H} P(h|D) \\ &= \arg\max_{h \in H} \frac{P(D|h)P(h)}{P(D)} \\ &= \arg\max_{h \in H} P(D|h)P(h) \end{aligned}$$

If assume $P(h_i) = P(h_j)$ then can further simplify, and choose the *Maximum likelihood* (ML) hypothesis

$$h_{ML} = \arg\max_{h_i \in H} P(D|h_i)$$

5.4 Neural Net Learning

- Motivated by studies of the brain.
- A network of "artificial neurons" that learns a function.
- Doesn't have clear decision rules like decision trees, but highly successful in many different applications. (e.g. face detection)
- Knowledge is represented in numeric form

Biological Neuron

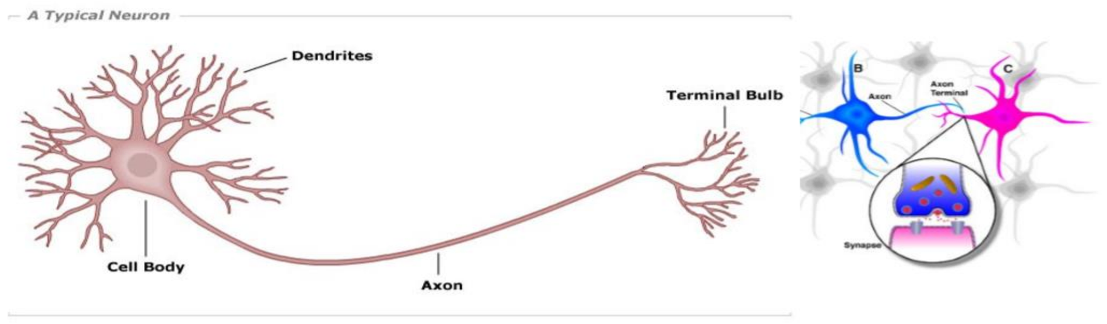
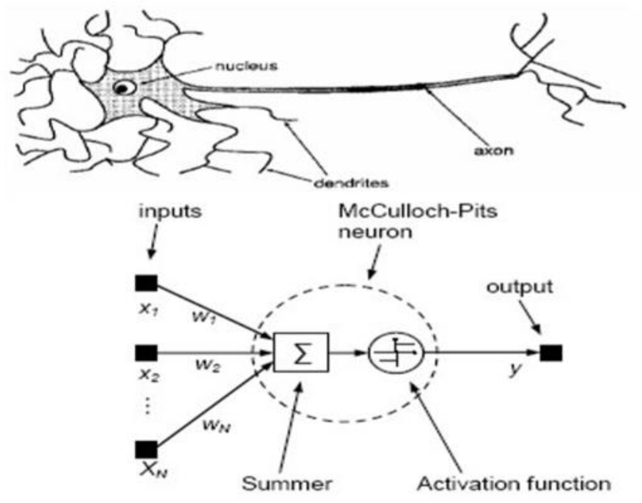


Figure: Biological Neuron

- Dendrites brings the input signals from other neurons
- Cell body gets the input signals from all dendrites and aggregates them. It then decides whether to send output signal through Axon or not
- Axon carries the impulse generated by the cell to other neurons.

- Axon is connected to dendrites of other neurons through synapse



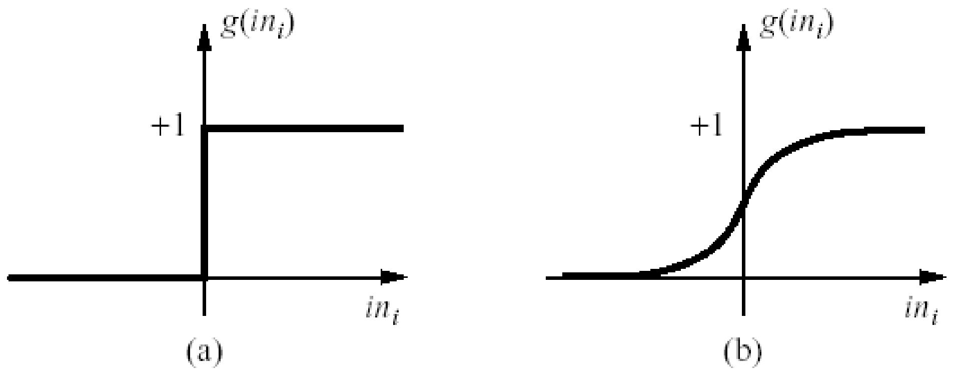
- NET = $X_1W_1+X_2W_2+....+X_nW_n$
- f (NET)= Out

For simple threshold function

f (NET)= Out = 1 if NET >=T
= 0 if NET < T

5.4.1 Activation Function

- Activation functions are mathematical equations that determine the output of a neural network.



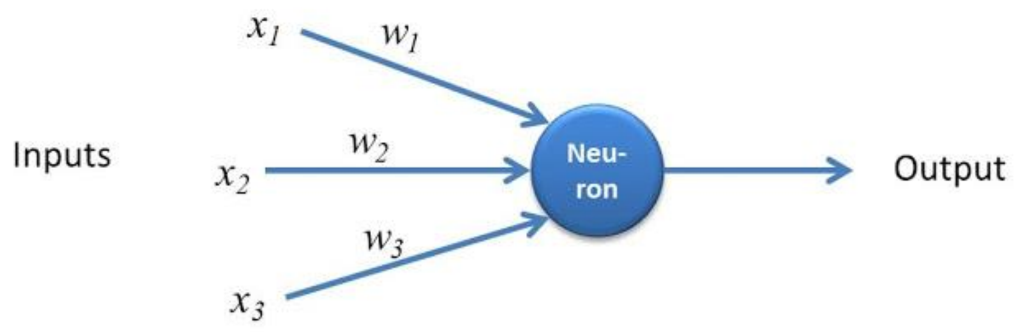
(a) is a step function or threshold function

(b) is a sigmoid function $1/(1+e^{-x})$

Changing the bias weight $W_{0,i}$ moves the threshold location

5.4.2 Perceptron

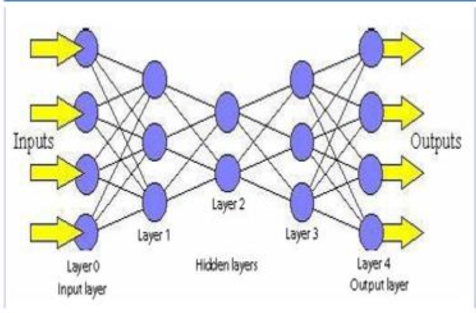
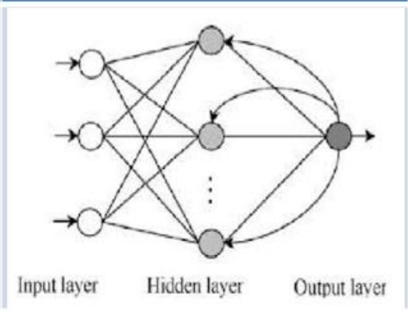
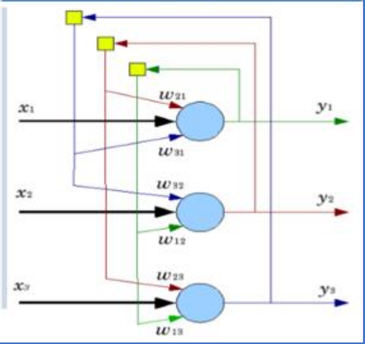
- The perceptron (or single-layer perceptron) is the simplest model of a neuron that illustrates how a neural network works.
- The perceptron is a machine learning algorithm developed in 1957 by Frank Rosenblatt and first implemented in IBM 704.



5.4.3 Architectures of NN

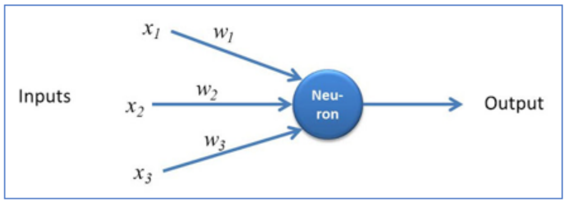
What do we mean by the architecture of NN?

- Way in which neurons are connected to together.

Feed Forward NN	Recurrent NN	Symmetrically connected NN
 Inputs Outputs Layer 0 Input layer Layer 1 Layer 2 Hidden layers Layer 3 Layer 4 Output layer		 x_1 x_2 x_3 w_{21} w_{31} w_{32} w_{12} w_{23} w_{13} y_1 y_2 y_3

How the Perceptron Works

- Example:
  - The perceptron has three inputs x1, x2 and x3 and one output.



$$output = \begin{cases} 0 \ \ if \sum_j w_j x_j < threshold \\ 1 \ \ if \sum_j w_j x_j > threshold \end{cases}$$

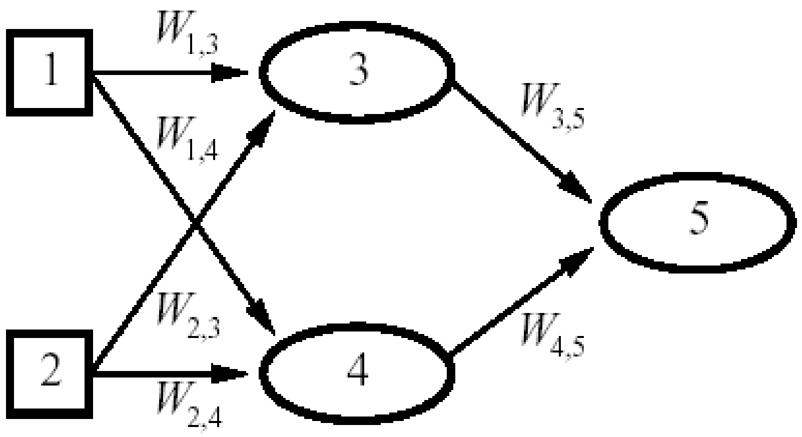
- Since the output of the perceptron could be either 0 or 1, this perceptron is an example of binary classifier

The Formula

Let's write out the formula that joins the inputs and the weights together to produce the output

$Output = w_1x_1 + w_2x_2 + w_3x_3$

Feed-forward example



Feed-forward network = a parameterized family of nonlinear functions:

$$\begin{aligned} a_5 &= g(W_{3,5} \cdot a_3 + W_{4,5} \cdot a_4) \\ &= g(W_{3,5} \cdot g(W_{1,3} \cdot a_1 + W_{2,3} \cdot a_2) + W_{4,5} \cdot g(W_{1,4} \cdot a_1 + W_{2,4} \cdot a_2)) \end{aligned}$$

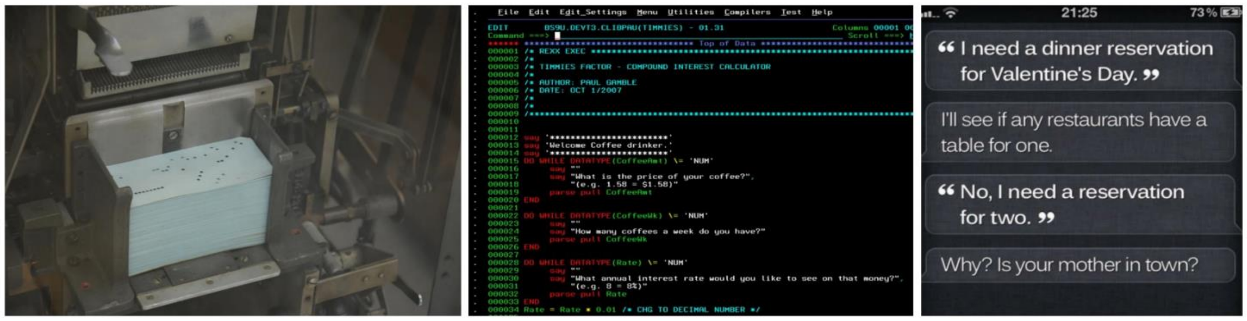
Adjusting weights changes the function: do learning this way!

CHAPTER 6

Communicating, Perceiving, and Acting

6.1 Natural Language Processing

Communication With Machines



```
EDIT       BS9U.DEVT3.CLIBPAU(TIMMIES) - 01.31                    Columns 00001 00
Command ===>                                                       Scroll ===>
****** ***************************** Top of Data ******************************
/* REXX EXEC *****************************************************************
/*
/* TIMMIES FACTOR - COMPOUND INTEREST CALCULATOR
/*
/* AUTHOR: PAUL GAMBLE
/* DATE: OCT 1/2007
/*
/*
/*****************************************************************************


say '************************'
say 'Welcome Coffee drinker.'
say '************************'
DO WHILE DATATYPE(CoffeeAmt) \= 'NUM'
      say ""
      say "What is the price of your coffee?",
          "(e.g. 1.58 = $1.58)"
      parse pull CoffeeAmt
END

DO WHILE DATATYPE(CoffeeWk) \= 'NUM'
      say ""
      say "How many coffees a week do you have?"
      parse pull CoffeeWk
END

DO WHILE DATATYPE(Rate) \= 'NUM'
      say ""
      say "What annual interest rate would you like to see on that money?",
          "(e.g. 8 = 8%)"
      parse pull Rate
END
Rate = Rate * 0.01 /* CHG TO DECIMAL NUMBER */
```

21:25 73%

“I need a dinner reservation for Valentine's Day.”

I'll see if any restaurants have a table for one.

“No, I need a reservation for two.”

Why? Is your mother in town?

Natural Language Processing or NLP refers to the branch of AI that gives machines the ability to read, understand and derive meaning from human languages. NLP combines the field of linguistics and computer science to decipher language structure and guidelines and to make models which can comprehend, break down and separate significant details from text and speech. NLP is required when you want an intelligent system like a robot to perform as per your instructions, when you want to hear decision from a dialogue based clinical expert system, etc.

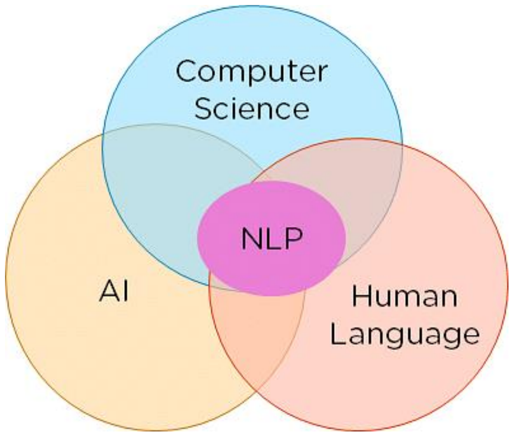


Figure 6.1: Disciplines that natural language processing comprises

Natural Language Processing (NLP) problem can divide into two tasks:

- ❖ **Processing written text**, using lexical, syntactic and semantic knowledge of the language as well as the required real-world information.
- ❖ **Processing spoken language**, using all the information needed above plus additional knowledge about phonology as well as enough added information to handle the further ambiguities that arise in speech.

6.1.1 Components of NLP

There are the following two components of NLP -

1. **Natural Language Understanding (NLU):** helps the machine to understand and analyse human language by extracting the metadata from content such as concepts, entities, keywords, emotion, relations, and semantic roles.

   NLU involves the following tasks -

   - It is used to map the given input into useful representation.
   - It is used to analyze different aspects of the language.

2. **Natural Language Generation (NLG);** acts as a translator that converts the computerized data into natural language representation. It mainly involves Text planning, Sentence planning, and Text Realization.

   Note: The NLU is more difficult than NLG.

Difference between NLU and NLG

NLU	**NLG**
NLU is the process of reading and interpreting language.	NLG is the process of writing or generating language.
It produces non-linguistic outputs from natural language inputs.	It produces constructing natural language outputs from non-linguistic inputs.

6.1.2 Applications of NLP

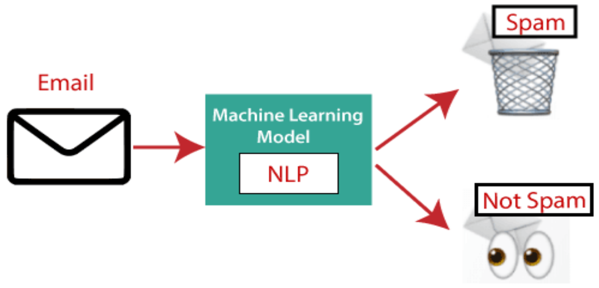
There are the following applications of NLP -

1. Question Answering

Question Answering focuses on building systems that automatically answer the questions asked by humans in a natural language.

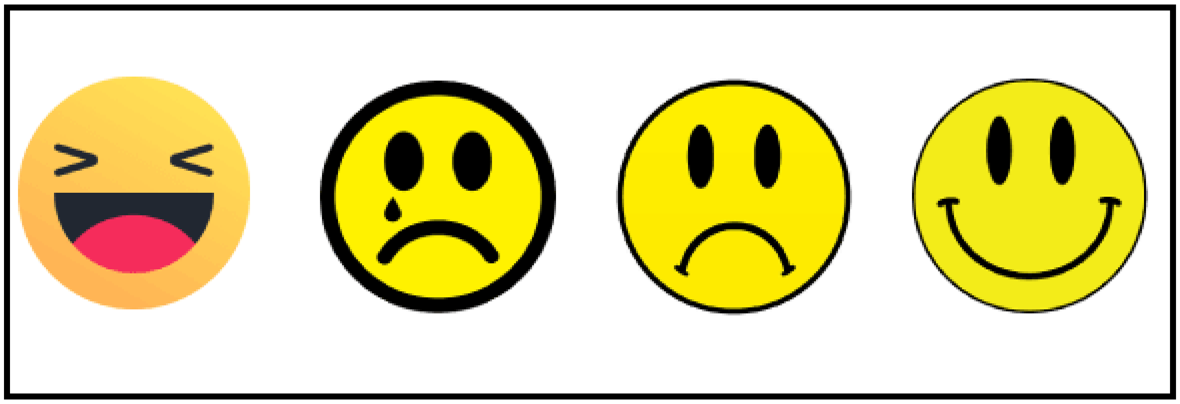
2. Spam Detection

Spam detection is used to detect unwanted e-mails getting to a user's inbox.



3. Sentiment Analysis

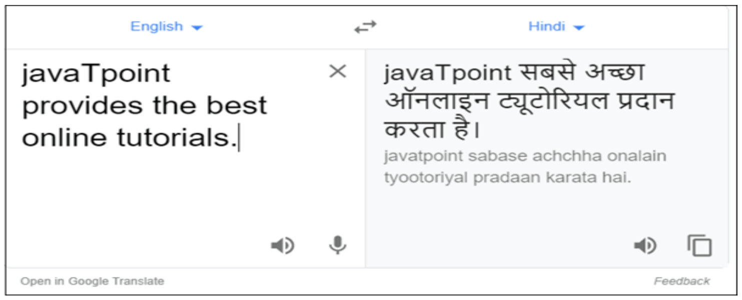
Sentiment Analysis is also known as **opinion mining**. It is used on the web to analyse the attitude, behavior, and emotional state of the sender. This application is implemented through a combination of NLP (Natural Language Processing) and statistics by assigning the values to the text (positive, negative, or natural), and identify the mood of the context (happy, sad, angry, etc.)



4. Machine Translation

Machine translation is used to translate text or speech from one natural language to another natural language.

Example: Google Translator



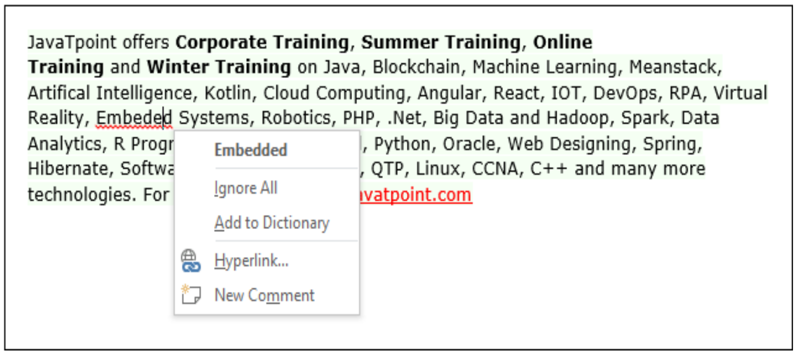
javaTpoint provides the best online tutorials.

javaTpoint सबसे अच्छा ऑनलाइन ट्यूटोरियल प्रदान करता है।

javatpoint sabase achchha onalain tyootoriyal pradaan karata hai.

5. Spelling correction

Spelling correction in NLP refers to detecting incorrect spellings and then correcting them.

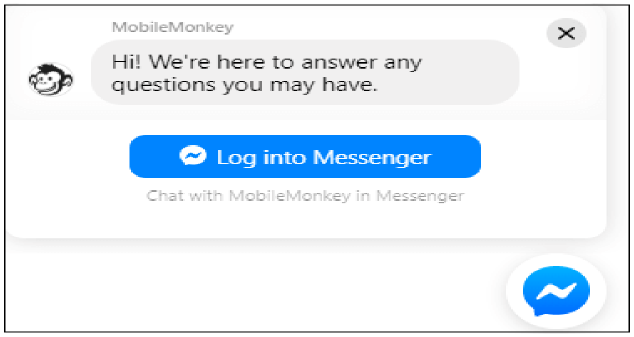

JavaTpoint offers **Corporate Training**, **Summer Training**, **Online Training** and **Winter Training** on Java, Blockchain, Machine Learning, Meanstack, Artifical Intelligence, Kotlin, Cloud Computing, Angular, React, IOT, DevOps, RPA, Virtual Reality, Embeded Systems, Robotics, PHP, .Net, Big Data and Hadoop, Spark, Data Analytics, R Progr, Python, Oracle, Web Designing, Spring, Hibernate, Softwa, QTP, Linux, CCNA, C++ and many more technologies. For avatpoint.com

6. Speech Recognition

Speech recognition is used for converting spoken words into text. It is used in applications, such as mobile, home automation, video recovery, dictating to Microsoft Word, voice biometrics, voice user interface, and so on.

7. Chatbot

Implementing the Chatbot is one of the important applications of NLP. It is used by many companies to provide the customer's chat services.


MobileMonkey

Hi! We're here to answer any questions you may have.

8. Information Extraction

Information extraction is one of the most important applications of NLP. It is used for extracting structured information from unstructured or semi-structured machine-readable documents.

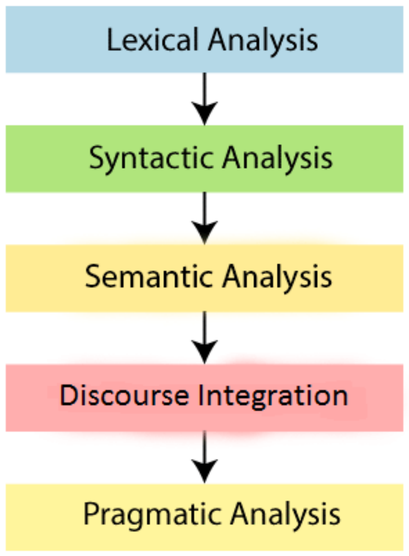
6.1.3 NLP Terminology

- **Phonology** – It is the study of organizing sound systematically.
- **Morphology** – It is a study of the construction of words from primitive meaningful units.
- **Morpheme** – It is a primitive unit of meaning in a language.
- **Syntax** – It refers to arranging words to make a sentence. It also involves determining the structural role of words in the sentence and in phrases.
- **Semantics** – It is concerned with the meaning of words and how to combine words into meaningful phrases and sentences.
- **Pragmatics** – It deals with using and understanding sentences in different situations and how the interpretation of the sentence is affected.
- **Discourse** – It deals with how the immediately preceding sentence can affect the interpretation of the next sentence.

- **World Knowledge** – It includes general knowledge about the world.

6.1.4 Phases/steps of NLP

There are the following five phases of NLP:



- **Lexical Analysis**
  - It involves identifying and analyzing the structure of words. The Lexicon of a language means the collection of words and phrases in a language. Lexical analysis is dividing the whole chunk of txt into paragraphs, sentences, and words.
  - Individual words analyzed into their components and non-word tokens such as punctuation separated from the words.
- **Syntactic Analysis (Parsing)**
  - It involves analysis of words in the sentence for grammar and arranging words in a manner that shows the relationship among the words.
  - Linear sequences of words transformed into structures that show how the words relate to each other.
  - Moreover, some word sequences may reject if they violate the language's rule for how words may combine.

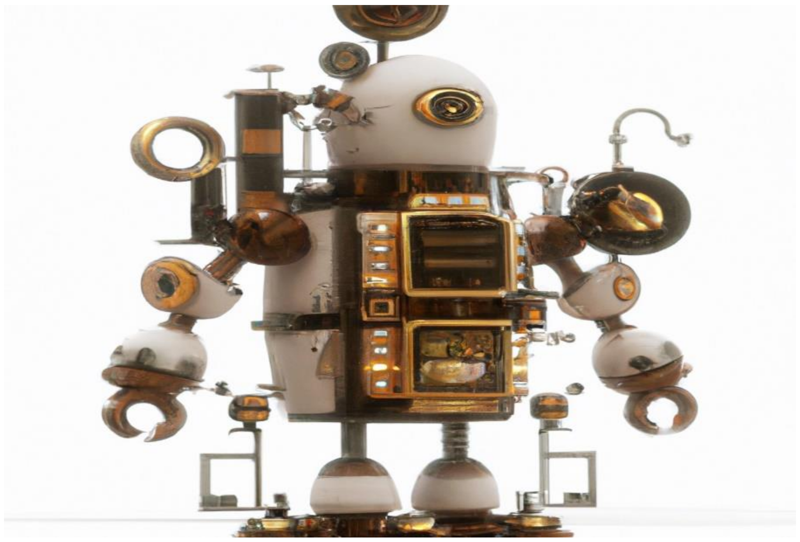
- **Semantic Analysis**
  - It draws the exact meaning or the dictionary meaning from the text. The text is checked for meaningfulness.
  - The structures created by the syntactic analyzer assigned meanings.
  - Also, a mapping is made between the syntactic structures and objects in the task domain.
  - Moreover, structures for which no such mapping is possible may reject.
- **Discourse integration**
  - The meaning of an individual sentence may depend on the sentences that precede it. And also, may influence the meanings of the sentences that follow it.
- **Pragmatic Analysis**
  - During this, the structure representing what said reinterpreted to determine what was actually meant. It involves deriving those aspects of language which require real-world knowledge.

6.2 Introduction to Robotics and Perception

6.2.1 What is Robotics?

A **robot** is a machine—especially one programmable by a computer—capable of carrying out a complex series of actions automatically. Robots may be constructed to evoke human form, but most robots are task-performing machines, designed with an emphasis on stark functionality, rather than expressive aesthetics.

The branch of technology that deals with the design, construction, operation, and application of robots,[as well as computer systems for their control, sensory feedback, and information processing. These technologies deal with automated machines that can take the place of humans in dangerous environments or manufacturing processes, or resemble humans in appearance, behavior, or cognition.



When you are given a task to perform, for example, place the chairs in the room into a circular pattern, you might proceed as follows. First, look around the room and assess the situation. How many chairs are there? Where are they located? Second, you might make plan for putting the chairs into the desired pattern: Go to the first chair, pick it up, and move it to its desired location; then, repeat this for the second chair, and so on. Finally, execute the plan, and arrange the chairs as desired. This basic strategy is often referred to as the Sense-Think-Act paradigm, and it forms the basis for the behavior of most all robotic systems.
In robotic systems, ***sensing*** might be performed using computer vision, touch sensors, laser range scanners, or any number of currently available sensing modalities. ***Thinking*** corresponds to the process of manipulating an internal model of the world, reasoning about which actions the robot could perform, and how these actions would change the state of the world. Finally, ***acting*** is the moment when the robot moves in, and interacts with, its environment.

For real robotic systems, it is almost never the case that a robot succeeds at performing a task by using a single instance of this sense-think-act process. Rather, these steps are typically performed iteratively. At each iteration, sensing updates the robot's understanding of the current situation; thinking is used to update or refine the current plan; actions are then executed to bring things a bit closer to the desired outcome. For this reason, one typically refers to the *sense-think-act loop*. The speed at which this loop is executed varies based on the task to be performed.

For a chess playing robot, a single iteration could take seconds or minutes (each iteration corresponding to a single move). For a self-driving car, a single iteration of sense-think-act might take milliseconds (or less). In the limit, as the time associated to a single iteration decreases to zero, we obtain continuous time systems. While the specific behavior of these various systems can be quite different, they are all nicely described in terms of the sense-think-act loop, and for this reason, we use this paradigm as the organizing structure for our development.

Action allows the robots to affect the world. Sensing allows robot to perceive the world.

6.2.2 Aspects of Robotics

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

6.2.3 Robotic Perception

In robotics, perception is understood as a system that endows the robot with the ability to perceive, comprehend, and reason about the surrounding environment. The key components of a perception system are essentially sensory data processing, data representation (environment modeling), and ML-based algorithms, as illustrated in figure 6.2.

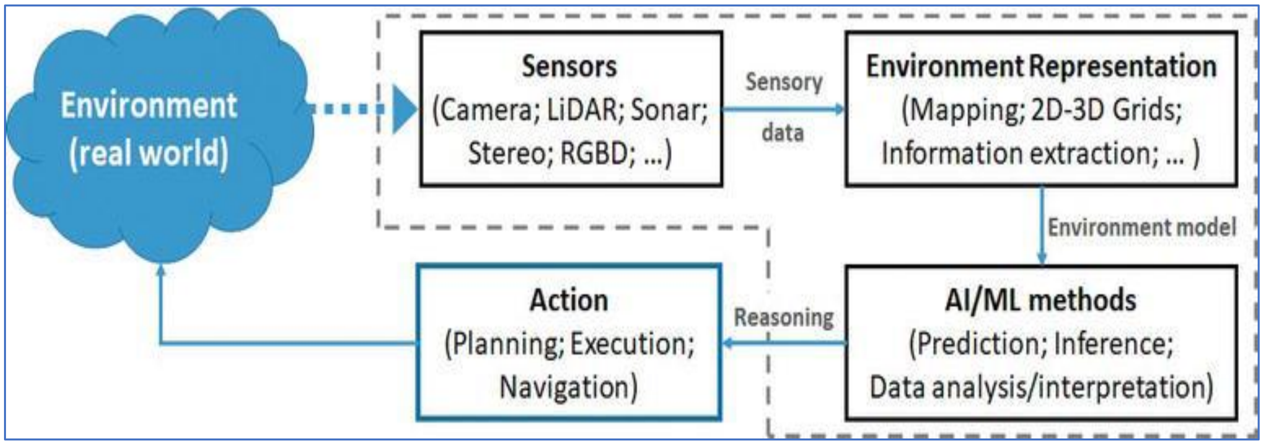

Environment (real world)
Sensors (Camera; LiDAR; Sonar; Stereo; RGBD; ...)
Sensory data
Environment Representation (Mapping; 2D-3D Grids; Information extraction; ...)
Environment model
AI/ML methods (Prediction; Inference; Data analysis/interpretation)
Reasoning
Action (Planning; Execution; Navigation)

Figure 6.2. Key modules of a typical robotic perception system: sensory data processing (focusing here on visual and range perception); data representations specific for the tasks at hand; algorithms for data analysis and interpretation (using AI/ML methods); and planning and execution of actions for robot-environment interaction.

Robotic perception is crucial for a robot to make decisions, plan, and operate in real-world environments, by means of numerous functionalities and operations from occupancy grid mapping to object detection. Some examples of robotic perception subareas, including autonomous robot-vehicles, are obstacle detection, object recognition, semantic place classification, 3D environment representation, gesture and voice recognition, activity classification, terrain classification, road detection, vehicle detection, pedestrian detection, object tracking, human detection, and environment change detection.

Nowadays, most of robotic perception systems use machine learning (ML) techniques, ranging from classical to deep-learning approaches. Machine learning for robotic perception can be in the form of unsupervised learning, or supervised classifiers using handcrafted features, or deep-learning neural networks (e.g., convolutional neural network (CNN)), or even a combination of multiple methods.

Regardless of the ML approach considered, data from sensor(s) are the key ingredient in robotic perception. Data can come from a single or multiple sensor, usually mounted onboard the robot, but can also come from the infrastructure or from another robot (e.g., cameras mounted on UAVs flying nearby). In multiple-sensors perception, either the same modality or multimodal, an efficient approach is usually necessary to combine and process data from the sensors before an ML method can be employed. Data alignment and calibration steps are necessary depending on the nature of the problem and the type of sensors used.

Text Book;

1. Stuart Russell and Peter Norvig (1995) Artificial Intelligence: A Modern Approach
2. Luger, G. (2002) Artificial Intelligence, 4th ed. Addison-Wesley.
3. Bratko, Ivan (1990) PROLOG Programming for Artificial Intelligence, 2nd edition.
4. Ginsberg, M.L. (1993) Essentials of Artificial Intelligence. Morgan Kaufman.