Artificial Intelligence (CSE)
UNIT-I

1.1 Introduction

Formal Definition of AI:

AI is a branch of computer science which is concerned with the study and creation of computer systems that exhibit

some form of intelligence

OR

those characteristics which we associate with intelligence in human behavior

AI is a broad area consisting of different fields, from machine vision, expert systems to the creation of machines that can "think". In order to classify machines as "thinking", it is necessary to define intelligence.

Intelligence:

Intelligence is a property of mind that encompasses many related mental abilities, such as the capabilities to

- reason
- plan
- solve problems
- think abstractly
- comprehend ideas and language and
- learn

Intelligent Systems:

An intelligent system is a system that can imitate, automate some intelligent behaviors of human being. Expert systems, intelligent agents and knowledge-based systems are examples of intelligent systems. Intelligent systems perform search and optimization along with learning capabilities. So they are technologically advanced machines that perceive and respond to the world around them. The field of intelligent systems also focuses on how these systems interact with human users in changing and dynamic physical and social environments.

Categories of AI System

- Systems that think like humans
- Systems that act like humans
- Systems that think rationally
- Systems that act rationally

Systems that think like humans

Most of the time it is a black box where we are not clear about our thought process. One has to know functioning of brain and its mechanism for possessing information. It is an area of cognitive science. The stimuli are converted into mental representation. Cognitive processes manipulate representation to build new representations that are used to generate actions. Neural network is a computing model for processing information similar to brain.

Systems that act like humans

The overall behavior of the system should be human like. It could be achieved by observation

Systems that think rationally

Such systems rely on logic rather than human to measure correctness. For thinking rationally or logically, logic formulas and theories are used for synthesizing outcomes.

For example,

given John is a human and all humans are mortal then one can conclude logically that John is mortal

Not all intelligent behavior are mediated by logical deliberation.

Systems that act rationally

Rational behavior means doing right thing. Even if method is illogical, the observed behavior must be rational.

Foundations of AI

Foundation of AI is based on

- Philosophy
- Mathematics
- Economics
- Neuroscience
- Control Theory
- Linguistics
- Computer Engineering
- Psychology

Philosophy:

- Can formal rules be used to draw valid conclusions?
- How does the mind arise from a physical brain?
- Where does knowledge come from?
- How does knowledge lead to action?

Mathematics:

- More formal logical methods
    - Boolean logic
    - Fuzzy logic
- Uncertainty
    - The basis for most modern approaches to handle uncertainty in AI applications can be handled by Probability theory, modal and temporal logics

Economics:

- How should we make decisions so as to maximize payoff?
- How should we do this when others may not go along?
- How should we do this when the payoff may be far in the future?

Neuroscience:

- How do the brain works?
    - Early studies (1824) relied on injured and abnormal people to understand what parts of brain work
    - More recent studies use accurate sensors to correlate brain activity to human thought
        - By monitoring individual neurons, monkeys can now control a computer mouse using thought alone
    - How close are we to have a mechanical brain?
        - Parallel computation, remapping, interconnections,….

Control Theory:

- Machines can modify their behavior in response to the environment (sense/action loop)
    - Water-flow regulator, steam engine governor, thermostat
- The theory of stable feedback systems (1894)
    - Build systems that transition from initial state to goal state with minimum energy
    - In 1950, control theory could only describe linear systems and AI largely rose as a response to this shortcoming

Linguistics:

- How does language relate to thought?
- Speech demonstrates so much of human intelligence
    - Analysis of human language reveals thought taking place in ways not understood in other settings
        - Children can create sentences they have never heard before
- Language and thought are believed to be tightly intertwined

Computer Engineering:

- How can we build an efficient computer ?

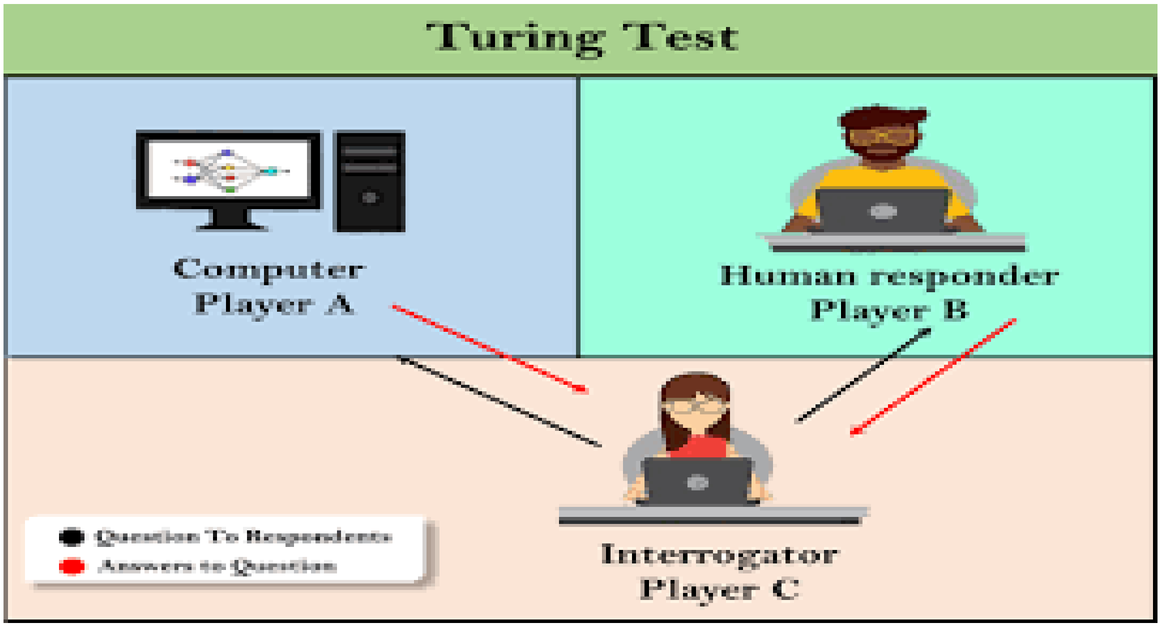
Psychology:

- How do humans and animals think and act ?

History of AI

Maturation / Gestation of Artificial Intelligence (1943-1952):

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



The birth of Artificial Intelligence (1952-1956):

- **Year 1955:** An Allen Newell and Herbert A. Simon created the "first artificial intelligence program which was named as "Logic Theorist". This program had proved 38 of 52 Mathematics theorems, and find new and more elegant proofs for some theorems.
- **Year 1956:** The word "Artificial Intelligence" first adopted by American Computer scientist John McCarthy at the Dartmouth Conference. For the first time, AI coined as an academic field.
- At that time high-level computer languages such as FORTRAN, LISP, or COBOL were invented. And the enthusiasm for AI was very high at that time.

The Golden years-Early enthusiasm (1956-1974):

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

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

The first AI winter (1974-1980):

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

A boom of AI (1980-1987):

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

The second AI winter (1987-1993)

- The duration between the years 1987 to 1993 was the second AI Winter duration.
- Again Investors and government stopped in funding for AI research as due to high cost but not efficient result. The expert system such as XCON was very cost effective.

The emergence of intelligent agents (1993-2011)

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

Deep learning, big data and artificial general intelligence (2011-present):

- **Year 2011:** In the year 2011, IBM's Watson won jeopardy, a quiz show, where it had to solve the complex questions as well as riddles. Watson had proved that it could understand natural language and can solve tricky questions quickly.
- **Year 2012:** Google has launched an Android app feature "Google now", which was able to provide information to the user as a prediction.
- **Year 2014:** In the year 2014, Chatbot "Eugene Goostman" won a competition in the infamous "Turing test."

Sub-areas of AI

Artificial Intelligence is having various sub fields in its domain. All the Sub-Fields can be distinguished as per various techniques :

Neural Networks

- Neural Networks are inspired by human brains and copies the working process of **human brains**. It is based on a collection of connected units or nodes called **artificial neurons** or **perceptrons**.

- The Objective of this approach was to solve the problems in the same way that a human brain does.

Vision

- In Artificial Intelligence Vision (Visioning Applications) means processing any image/video sources to extract meaningful information and take action based on that.
- In this field of artificial Intelligence we have also developed such kind of robots which are acquiring human activities within some days or sometimes some hours and train themselves . For e.g. object recognition, image understanding , Playing Robots etc.

Machine Learning

- The capability of Artificial Intelligence systems to learn by extracting patterns from data is known as **Machine Learning**.
- It is an approach or subset of Artificial Intelligence that is based on the idea that machines can be given access to data along with the ability to learn from it.

Speech Processing / Speech Recognition

- Speech Processing / Recognition is the ability of a computer and a program to identify words and phrases in the spoken language and convert them to machine readable format.
- The real life examples of Speech processing are Google Assistant ,Amazon Alexa and Apple’s Siri Application etc.

Robotics

- Robots are the **artificial agents** which behaves like human and build for the purpose of manipulating the objects by perceiving, picking, moving, modifying the physical properties of object, or to have an effect thereby freeing manpower from doing repetitive functions without getting bored, distracted, or exhausted.

Applications

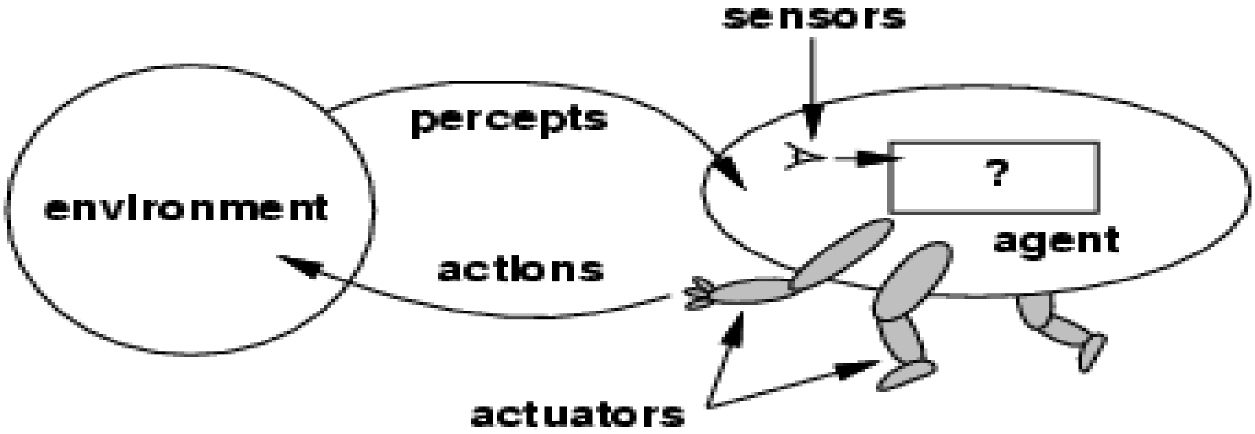
Some of the applications are given below:

- **Business :** Financial strategies, give advice
- **Engineering:** check design, offer suggestions to create new product
- **Manufacturing:** Assembly, inspection & maintenance
- **Mining:** used when conditions are dangerous
- **Hospital :** monitoring, diagnosing & prescribing
- **Education :** In teaching
- **Household :** Advice on cooking, shopping etc.
- **Farming :** prune trees & selectively harvest mixed crops.

Structure of Agents

Agents

Definition: An agent perceives its environment via sensors and acts upon that environment through its actuators



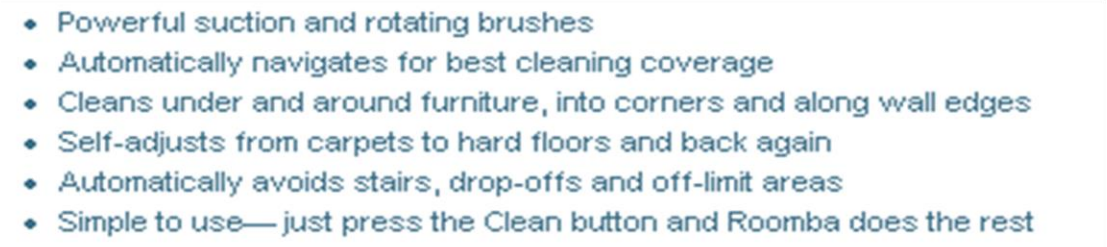
agent = architecture + program

Example: Vacuum Cleaner World

iRobot Roomba® 400
Vacuum Cleaning Robot

iRobot Corporation

Founder Rodney Brooks (MIT)



- Powerful suction and rotating brushes
- Automatically navigates for best cleaning coverage
- Cleans under and around furniture, into corners and along wall edges
- Self-adjusts from carpets to hard floors and back again
- Automatically avoids stairs, drop-offs and off-limit areas
- Simple to use— just press the Clean button and Roomba does the rest

Rational Agents:

An agent should strive to "do the right thing", based on what:

- it can perceive and
- the actions it can perform.

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 maximizes its performance measure (in expectation)** given the evidence provided by the percept sequence and whatever built-in knowledge the agent has.

Fully observable / Partially observable Environments

- If an agent's sensors give it access to the complete state of the environment needed to choose an action, the environment is fully observable.
    - (e.g. Chess)
- An environment might be partially observable because of noisy and inaccurate sensors or because parts of the state are simply missing from the sensor data
    - (e.g. Kriegspiel chess)

Characterizing a Task Environment

PEAS: Performance measure, Environment, Actuators, Sensors

Example: the task of designing a self-driving car

- **Performance measure** Safe, fast, legal, comfortable trip
- **Environment** Roads, other traffic, pedestrians
- **Actuators** Steering wheel, accelerator, brake, signal, horn
- **Sensors** Cameras, LIDAR (light/radar), speedometer, GPS, odometer**,** engine sensors, keyboard

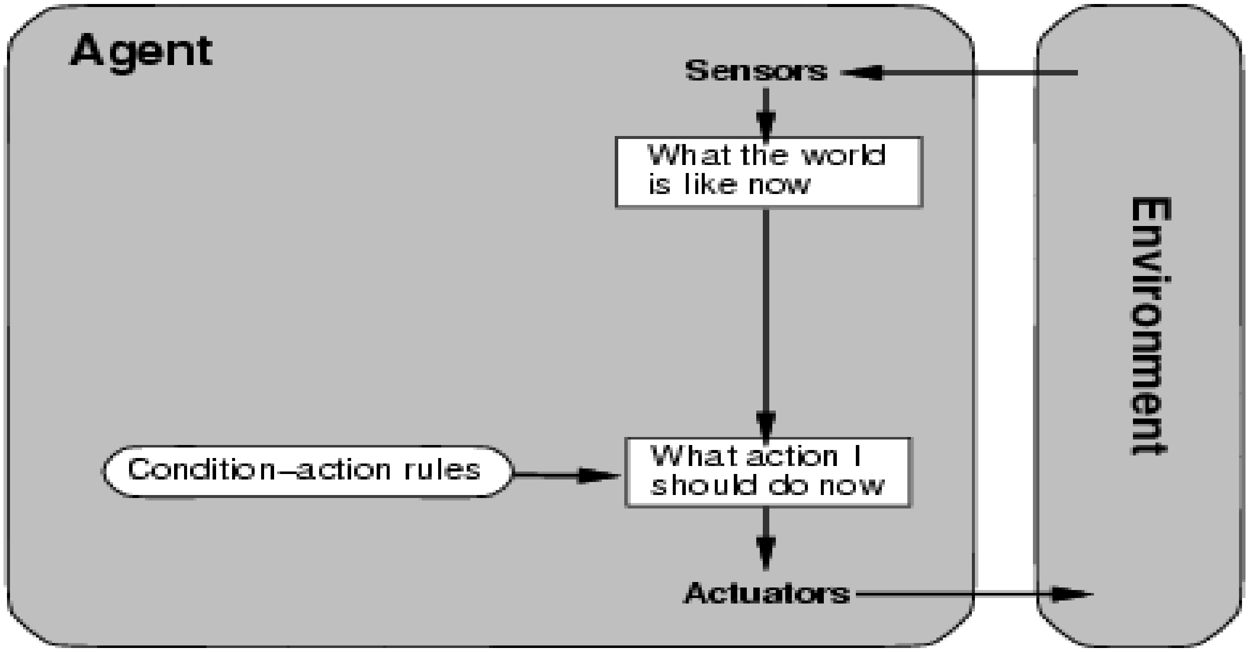
Types of Agents

1. Simple reflex agents
2. Model based reflex agents
3. Goal based agents
4. Utility based agents
5. Learning agents

1. Simple reflex agents:

- Agents do not have memory of past world states or percepts. So, actions depend solely on current percept.
- In this agent Action becomes a "reflex."
- So it uses condition-action rules.
- Agent selects actions on the basis of *current* percept only.

Ex: If tail-light of car in front is red, then brake.



Schematic diagram of Simple reflex agent

```
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
```

Example: The agent program for a simple reflex agent in the two-state vacuum environment.

```
function REFLEX-VACUUM-AGENT([location,status]) returns an action
if status = Dirty then return Suck
else if location = A then return Right
else if location = B then return Left
```

2. Model-based reflex agents

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

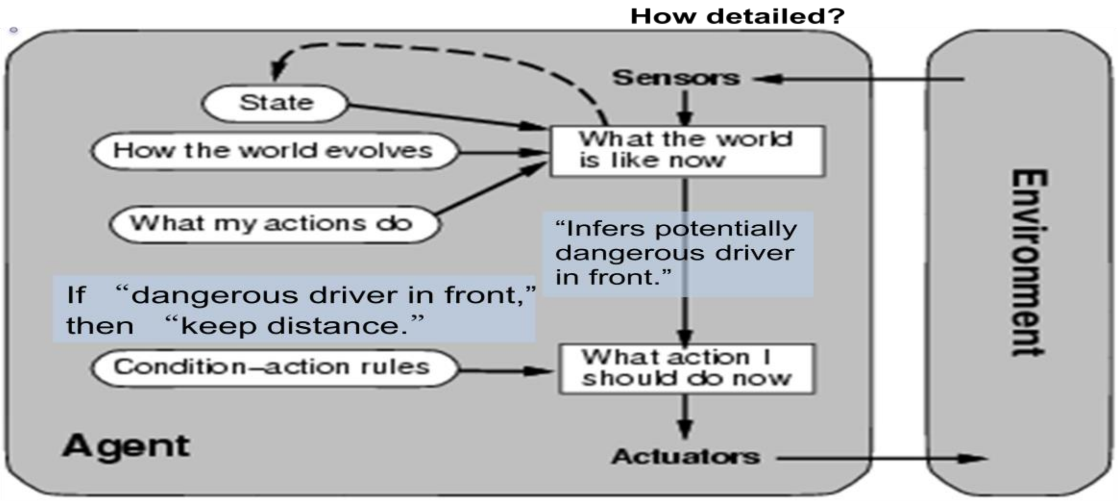
Model − knowledge about “how the things happen in the world”.

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

Updating the state requires the information about −

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

Model-based reflex agents


How detailed?
Sensors
State
How the world evolves
What my actions do
What the world is like now
"Infers potentially dangerous driver in front."
If "dangerous driver in front," then "keep distance."
Condition–action rules
What action I should do now
Agent
Actuators
Environment

Schematic diagram of Model based reflex agent

```
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
```

An example: Brooks' Subsumption Architecture

- Main idea: build complex, intelligent robots by decomposing behaviors into a hierarchy of skills, each defining a percept-action cycle for one very specific task.
- Examples: collision avoidance, exploring, recognizing doorways, etc.
- Each behavior is modeled by a finite-state machine with a few states (though each state may correspond to a complex function or module; provides internal state to the agent).
- Behaviors are loosely coupled via asynchronous interactions.

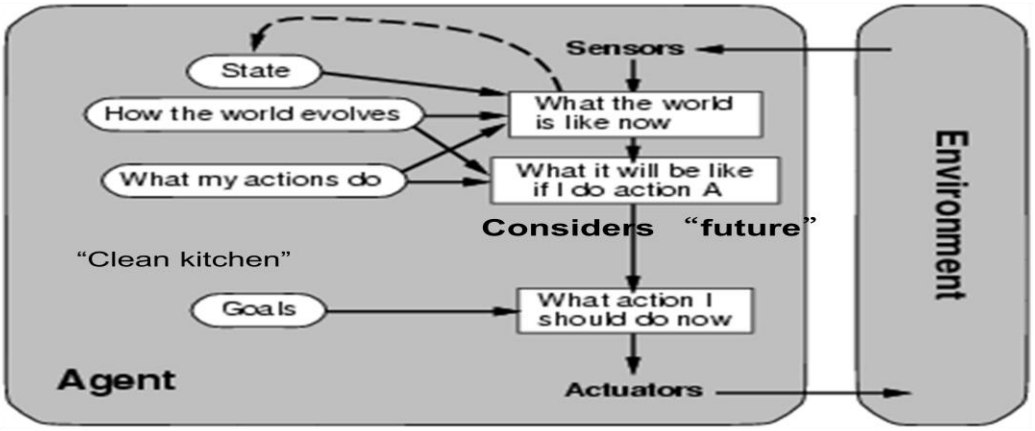
3. Goal-based agents

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

Goal − It is the description of desirable situations.

Module:
Problem Solving

Goal-based agents



Agent keeps track of the world state as well as set of goals it's trying to achieve: chooses actions that will (eventually) lead to the goal(s).
More flexible than reflex agents → may involve search and planning

4. Utility-based agents

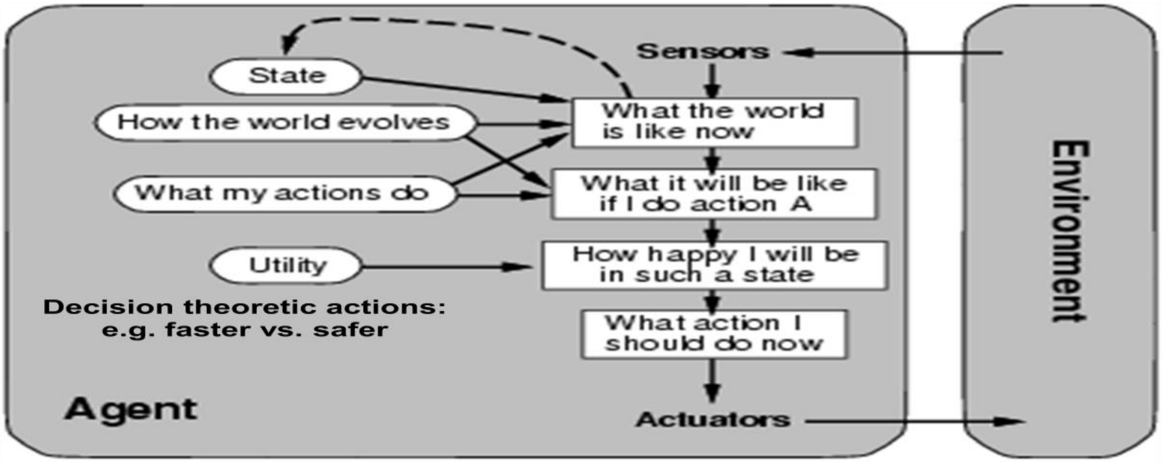
They choose actions based on a preference (utility) for each state.

Goals are inadequate when −

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

Module:
Decision Making

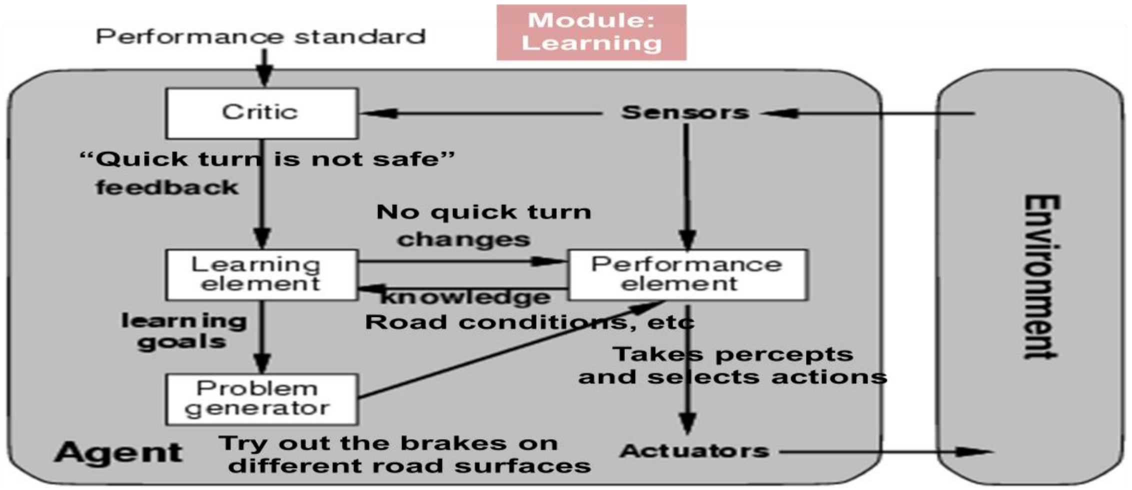
Utility-based agents


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
Utility
How happy I will be in such a state
Decision theoretic actions: e.g. faster vs. safer
What action I should do now
Agent
Actuators
Environment

5. Learning agents

- Learning agents are such agents which adapts and improve over time.
- More complicated when agent needs to learn utility information: Reinforcement learning

Learning agents



Overview of Structure of Agents:

An agent perceives and acts in an environment, has an architecture, and is implemented by an agent program.

A rational agent always chooses the action which maximizes its expected performance, given its percept sequence so far.

An autonomous agent uses its own experience rather than built-in knowledge of the environment by the designer.

An agent program maps from percept to action and updates its internal state.

- **Simple reflex agents**
  are based on condition-action rules, implemented with an appropriate production system. They are stateless devices which do not have memory of past world states.
- **Agents with memory - Model-based reflex agents**
  have internal state, which is used to keep track of past states of the world.
- **Agents with goals – Goal-based agents**
  are agents that, in addition to state information, have goal information that describes desirable situations. Agents of this kind take future events into consideration.
- **Utility-based agents**
  base their decisions on classic axiomatic utility theory in order to act rationally.
- **Learning agents**
  they have the ability to improve performance through learning.

Representing knowledge is important for successful agent design.

1.2 Problem Solving

Well defined Problems and Solutions:

A problem can be defined formally by five components:

Initial State:

The state where agents starts to perform the search to reach the goal state

Actions:

Set of applicable actions perform in state S.

Transition model:

What each actions does

Goal Test:

Which determines whether a given state is a goal state or not.

Path Cost:

Assigns a numeric cost to each path

State Space

- Together the initial state, actions and transition model implicitly define the state space of the problem – the set of all states reachable from the initial state by any sequence of actions.
- The state space forms a directed network or graph in which the nodes are states and the links between nodes are actions.
- A path in the state space is a sequence of states connected by a sequence of actions.

Example: Travelling in Romania

Scenario

- On holiday in Romania; currently in Arad
- Flight leaves tomorrow from Bucharest

Formulate Goal : Be in Bucharest

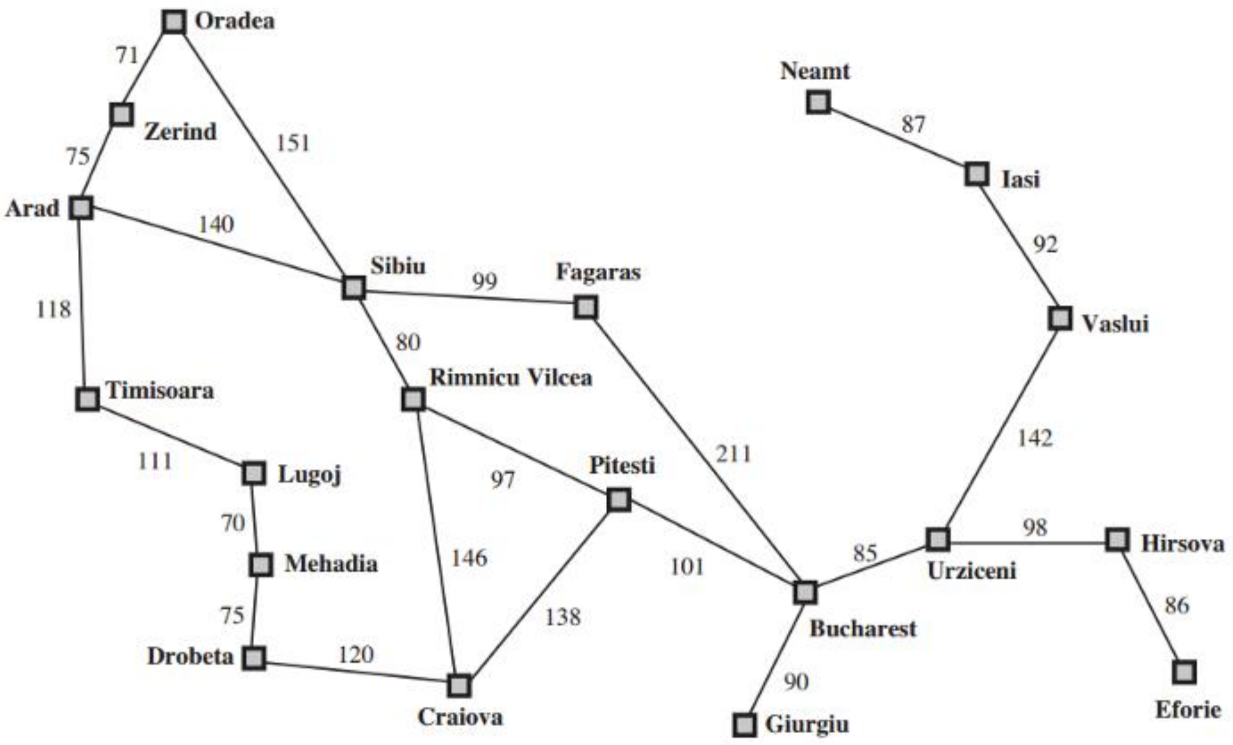
Formulate problem

***States*:** various cities

***Actions*:** drive between cities

Find Solution: Appropriate sequence of cities

e.g.: Arad, Sibiu, Fagaras, Bucharest



Formulating Problems:

- We proposed a formulation of the problem of getting to Bucharest in terms of the initial state, actions, transition model, goal test and path cost.
- This formulation seems reasonable, but it is still a model—an abstract mathematical description.
- We left out many things while travelling like the travelling companions, the current radio program, the scenery out of the window, the proximity of law enforcement officers, the distance to the next rest stop, the condition of the road, the weather and so on.
- These details are irrelevant to the problem of finding a route to Bucharest. The process of removing detail from a representation is called abstraction.

Problem types

1. Single –state problem
2. Multiple –state problem
3. Contingency problem.

Single –state problem

- observable (at least initial state)
- deterministic
- static
- discrete

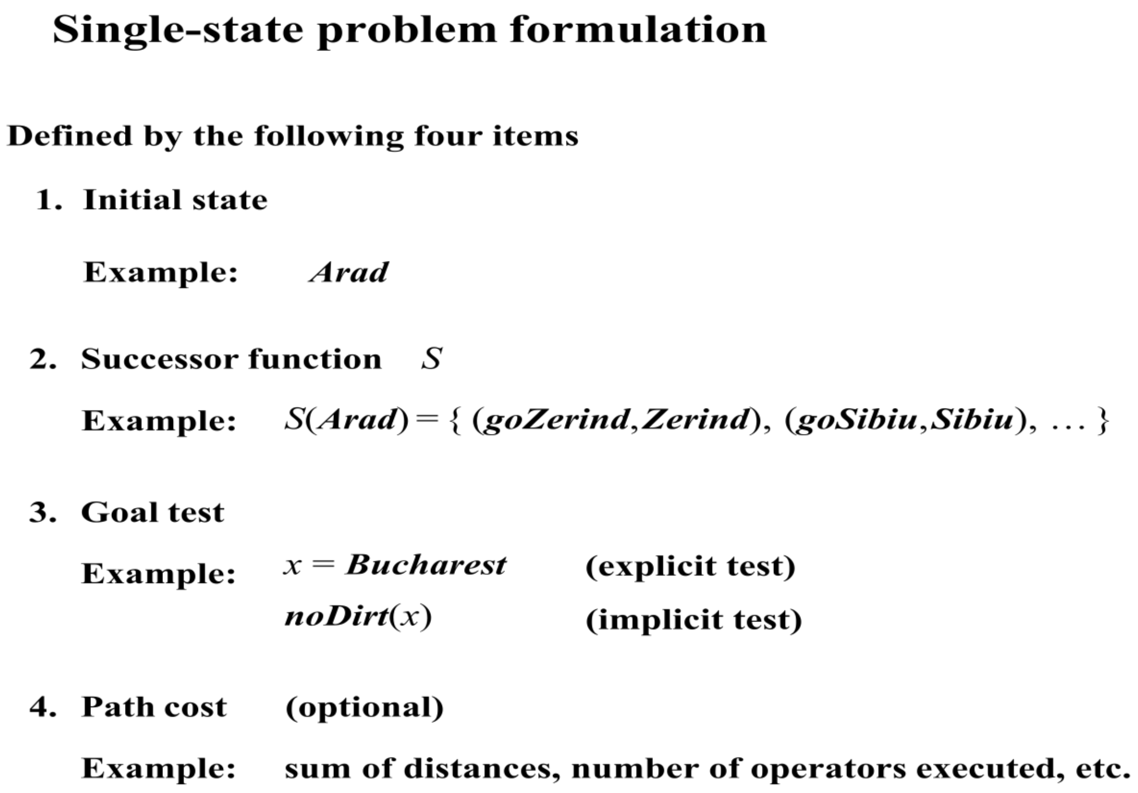
Multiple –state problem

- partially observable (initial state not observable)
- deterministic

- static
- discrete

Contingency problem

- partially observable (initial state not observable)
- non-deterministic



Single-state problem formulation

Defined by the following four items

1. **Initial state**

   Example: *Arad*

2. **Successor function** S

   Example: $S(\textit{Arad}) = \{ (\textit{goZerind}, \textit{Zerind}), (\textit{goSibiu}, \textit{Sibiu}), \ldots \}$

3. **Goal test**

   Example: $x = \textit{Bucharest}$ **(explicit test)**

   $\textit{noDirt}(x)$ **(implicit test)**

4. **Path cost** **(optional)**

   Example: **sum of distances, number of operators executed, etc.**

Solution :
A sequence of operators leading from the initial state to a goal state

Selecting a state space

Abstraction
Real world is absurdly complex
State space must be abstracted for problem solving

(Abstract) state   Set of real states

(Abstract) operator
Complex combination of real actions

Example: *Arad* → *Zerind* represents complex set of possible routes

(Abstract) solution
Set of real paths that are solutions in the real world

Problems on State Space representation

Example: The 8-puzzle

- It is also called as **N puzzle problem** or **sliding puzzle problem.**
- **N-puzzle** that consists of N tiles (N+1 titles with an empty tile) where N can be 8, 15, 24 and so on.
- In our example **N = 8.** (that is **square root of (8+1) = 3 rows and 3 columns**).
- In the same way, if we have N = 15, 24 in this way, then they have Row and columns as follow **(square root of (N+1) rows and square root of (N+1) columns).**
- That is if **N=15** than number of rows and columns= 4, and if **N= 24** number of rows and columns= 5.
- So, basically in these types of problems we have given a **initial state or initial configuration (Start state) and a Goal state or Goal Configuration.**
- Here we are solving a problem **of 8 puzzle** that is **a 3x3 matrix.**

Start State

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

Goal State

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

States: Integer location of tiles

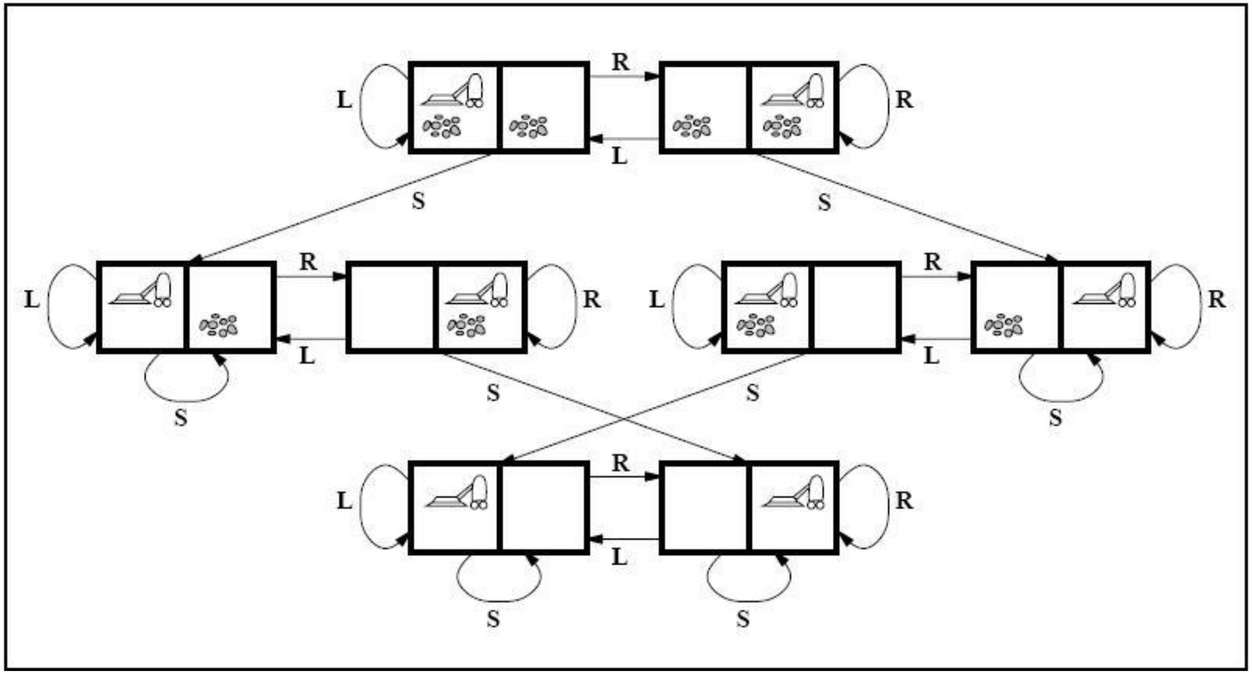
Actions: left, right, up, down

Goal Test: = goal state?

Path Cost: 1 per move

8 Puzzle has 9! / 2 = 181440 states

Example: Vacuum-cleaner world state space graph



States: Integer dirt and robot locations

Actions: left, right, suck, noOp

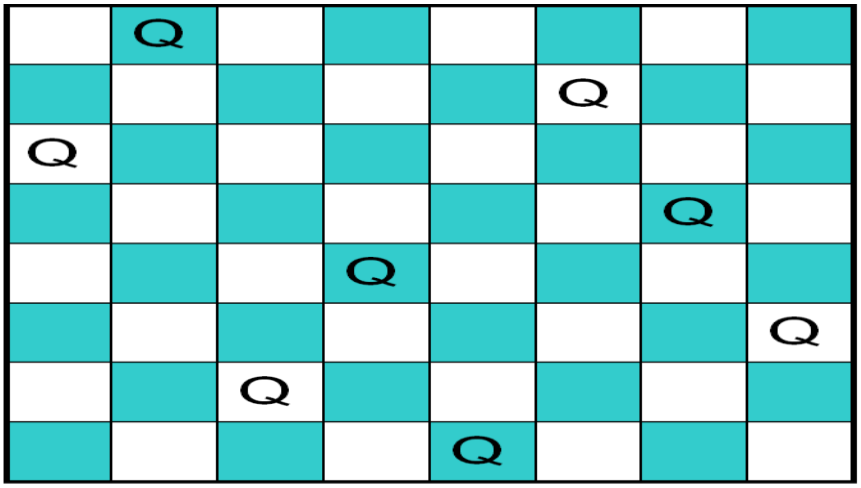
Goal Test: not dirty?

Path Cost: 1 per operation (0 for noOp)

Example: Eight Queens

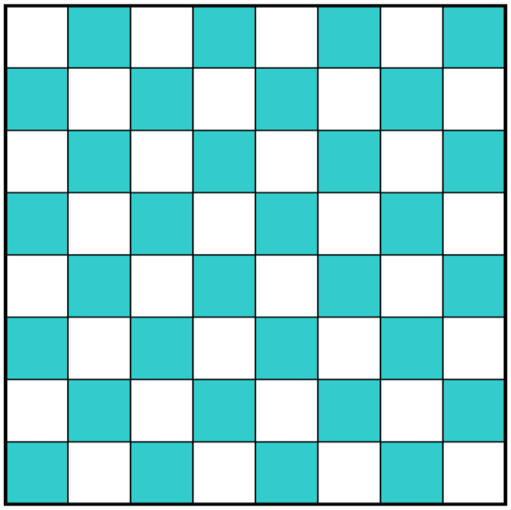
- Place eight queens on a chess board such that no queen can attack another queen
- No path cost because only the final state counts!
- Incremental formulations
- Complete state formulations

Solutions:

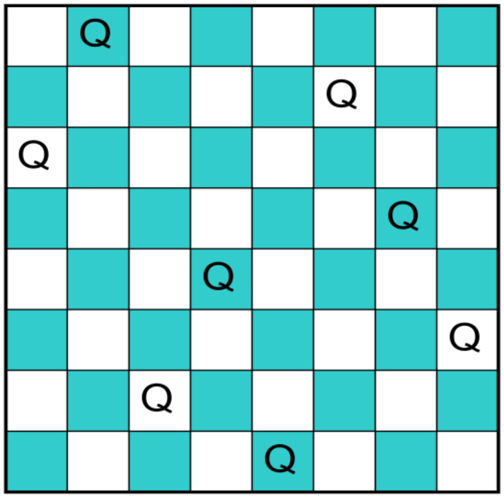


Eight Queens Problem Formulation 1:

- **States:**
  - Any arrangement of 0 to 8 queens on the board
- **Initial state:**
  - No queens on the board
- **Successor function:**
  - Add a queen to an empty square
- **Goal Test:**
  - 8 queens on the board and none are attacked
- $64*63*\ldots*57 = 1.8*10^{14}$ possible sequences



Empty Board

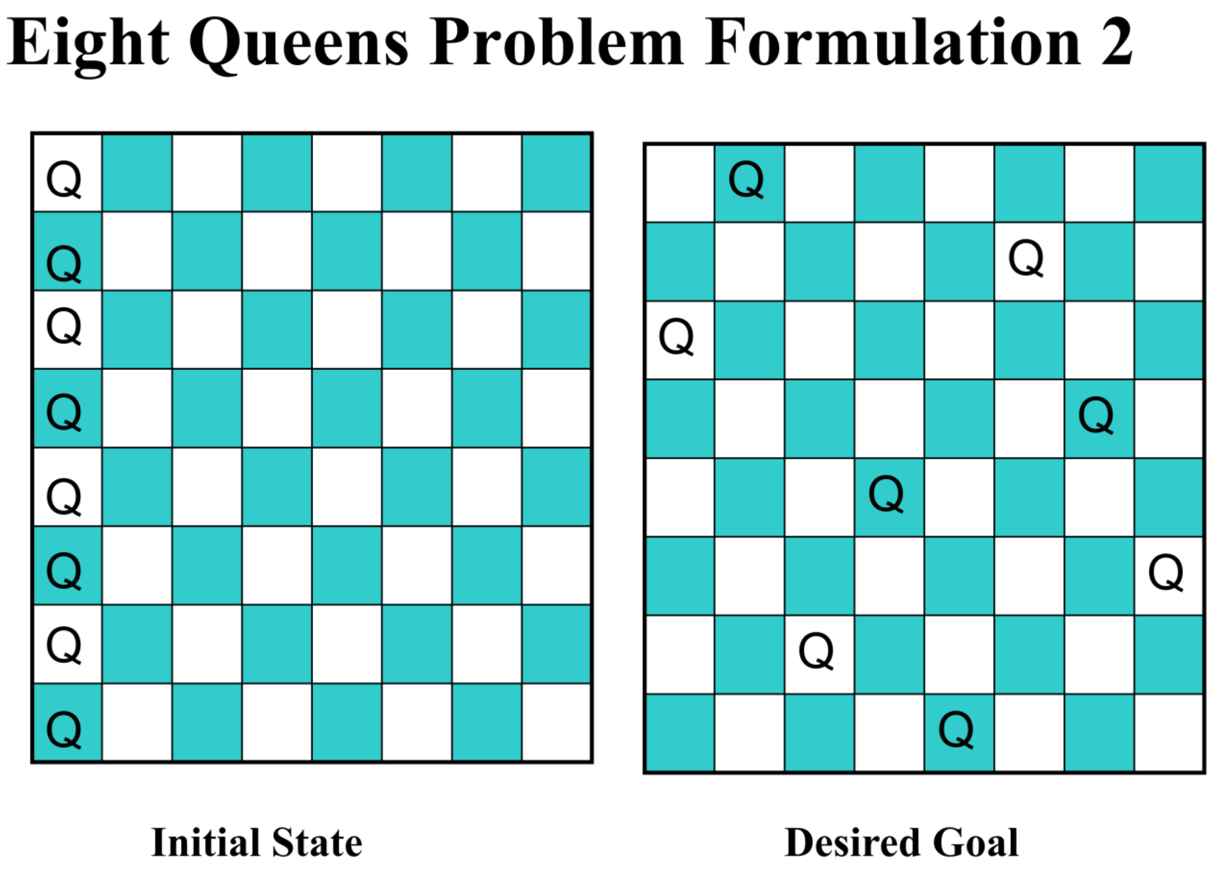


Desired Goal

Eight Queens Problem Formulation 2:

- **States:**
  - Any arrangement of 8 queens on the board
- **Initial state:**
  - All queens are at column 1

- **Successor function:**
    - Change the position of any one queen
- **Goal Test:**
    - 8 queens on the board and none are attacked



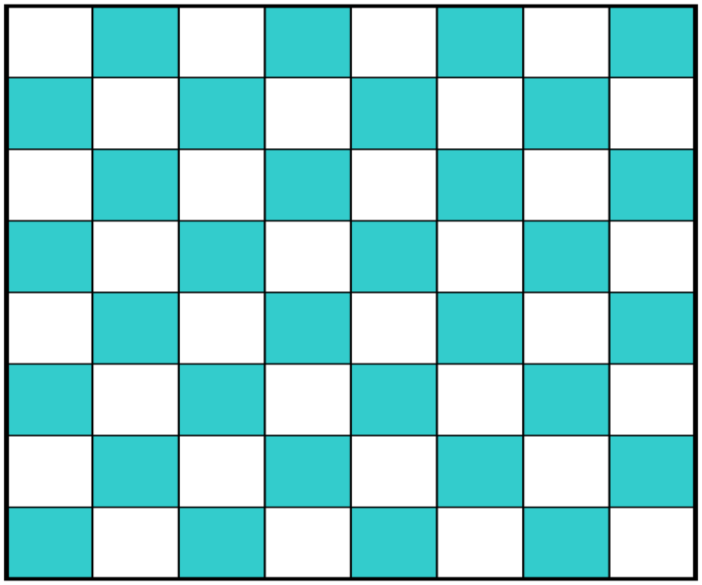
Eight Queens Problem Formulation 2

Initial State

Desired Goal

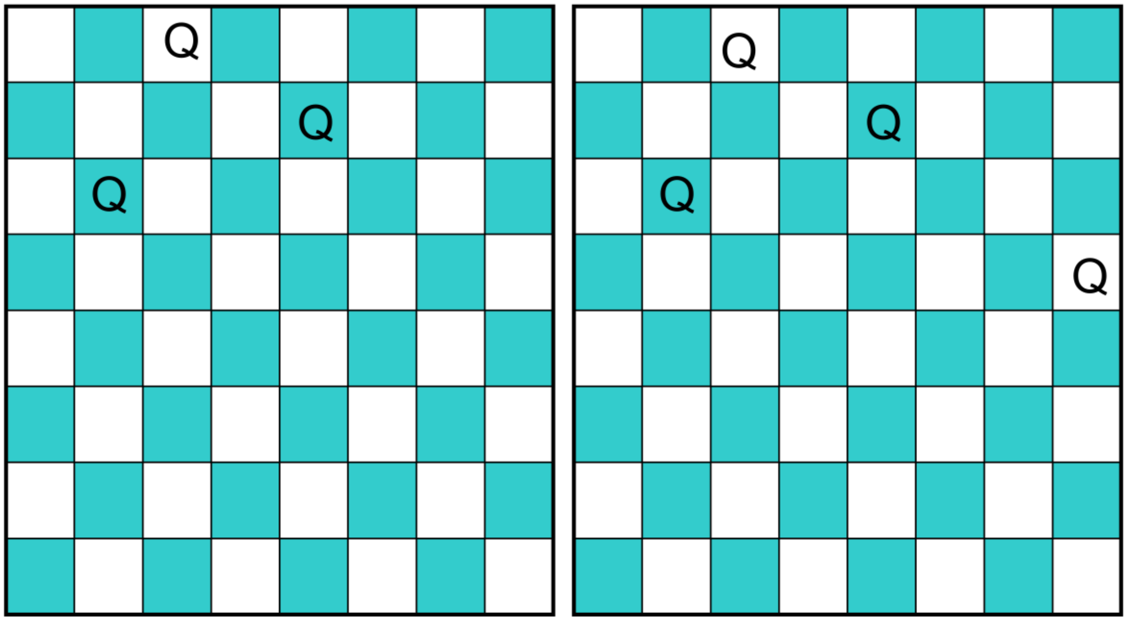
Eight Queens Problem Formulation 3:

- **States:**
    - Any arrangement of k queens in the first k rows such that none are attacked
- **Initial state:**
    - No queens on the board
- **Successor function:**
    - Add a queen to the (k+1)th row so that none are attacked
- **Goal Test:**
    - 8 queens on the board and none are attacked



Initial State

Eight Queens Problem Formulation 3



(k)th row , here k=3

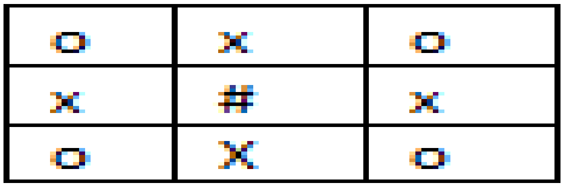
Adding a queen at (k+1)th row

Solving 8 Puzzle Problem :

The puzzle can be solved by moving the tiles one by one in the single empty space and thus achieving the Goal state.

Rules of solving puzzle

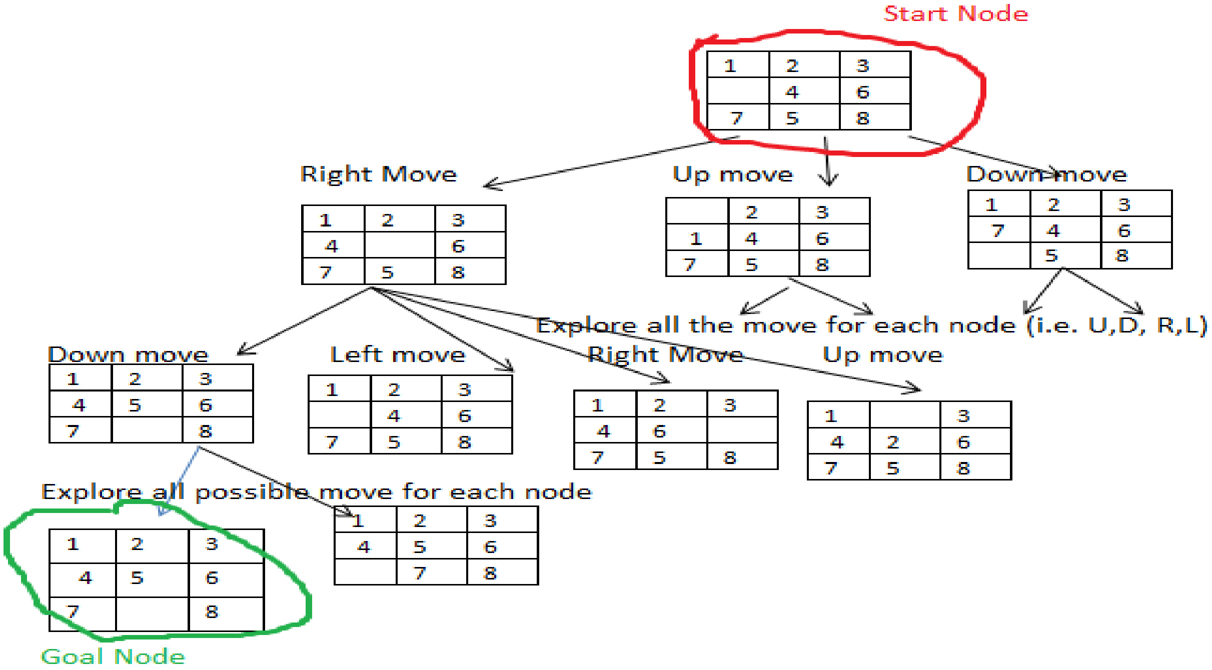
- Instead of moving the tiles in the empty space we can visualize moving the empty space in place of the tile.
- The empty space can only move in four directions (Movement of empty space)
- Up
- Down
- Right
- Left
- The empty space cannot move diagonally and can take only one step at a time.



o	x	o
x	#	x
o	x	o

- **o- Position** total possible moves are **(2)**,
- **x - position** total possible moves are **(3)** and
- **#-position** total possible moves **are (4)**

Let's solve the problem without **Heuristic Search** that is **Uninformed Search or Blind Search (Breadth First Search and Depth First Search)**


Start Node
Right Move
Up move
Down move
Explore all the move for each node (i.e. U,D, R,L)
Down move
Left move
Right Move
Up move
Explore all possible move for each node
Goal Node

__**Breadth First Search to solve Eight puzzle problem**__

1.3 Search Strategies

Search Algorithm

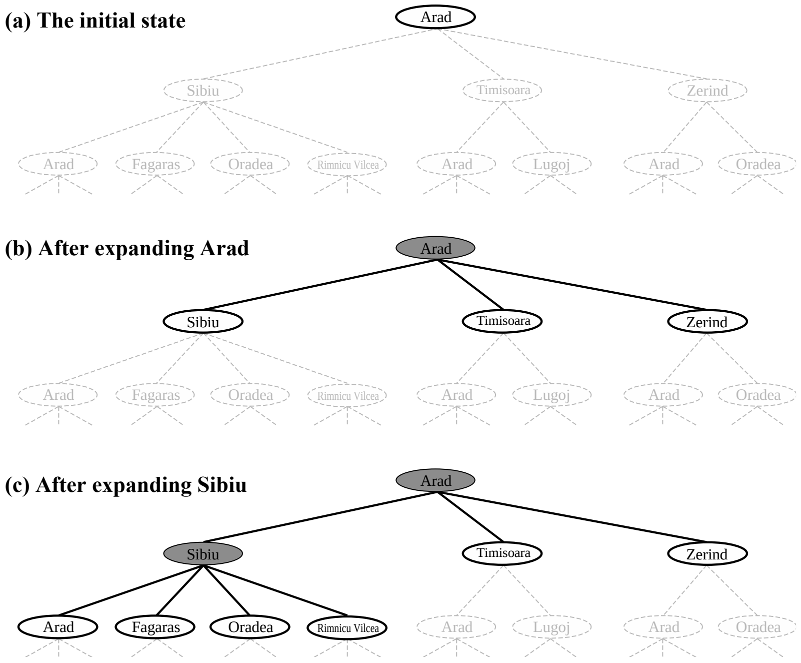
Search Algorithm teaches computers to “act rationally” by achieving a certain goal with a certain input value

Search: Searching is a step by step procedure to solve a search-problem in a given search space. A search problem can have three main factors:

- **Search Space:** Search space represents a set of possible solutions, which a system may have.
- **Start State:** It is a state from where agent begins **the search**.
- **Goal test:** It is a function which observe the current state and returns whether the goal state is achieved or not.

Search tree: A tree representation of search problem is called Search tree. The root of the search tree is the root node which is corresponding to the initial state.

Search Trees



(a) The initial state

(b) After expanding Arad

(c) After expanding Sibiu

Partial search trees for finding a route from Arad to Bucharest

```
function TREE-SEARCH(problem) returns a solution, or failure
    initialize the frontier using the initial state of problem
    loop do
        if the frontier is empty then return failure
        choose a leaf node and remove it from the frontier
        if the node contains a goal state then return the corresponding solution
```

```
      expand the chosen node, adding the resulting nodes to the frontier
function GRAPH-SEARCH(problem) returns a solution, or failure
  initialize the frontier using the initial state of problem
  initialize the explored set to be empty
  loop do
    if the frontier is empty then return failure
    choose a leaf node and remove it from the frontier
    if the node contains a goal state then return the corresponding solution
    add the node to the explored set
    expand the chosen node, adding the resulting nodes to the frontier
      only if not in the frontier or explored set
```

An informal description of the general tree-search and graph-search algorithms.

Properties of Search Algorithms:

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

- **Completeness:** A search algorithm is said to be complete if it guarantees to return a solution if at least any solution exists for any random input.
- **Optimality:** If a solution found for an algorithm is guaranteed to be the best solution (lowest path cost) among all other solutions, then such a solution for is said to be an optimal solution.
- **Time Complexity:** Time complexity is a measure of time for an algorithm to complete its task.
- **Space Complexity:** It is the maximum storage space required at any point during the search, as the complexity of the problem.

Types of search algorithms:

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

Uninformed/Blind Search:

- The uninformed search does not contain any domain knowledge such as closeness, the location of the goal. It operates in a brute-force way as it only includes information about how to traverse the tree and how to identify leaf and goal nodes.
- Uninformed search applies a way in which search tree is searched without any information about the search space like initial state operators and test for the goal, so it is also called blind search. It examines each node of the tree until it achieves the goal node.
- So Distance to goal not taken into account
- Ex: DFS, BFS, Iterative deepening DFS

Informed Search:

- Informed search algorithms use domain knowledge. In an informed search, problem information is available which can guide the search. Informed search strategies can find a solution more efficiently than an uninformed search strategy. Informed search is also called a Heuristic search.

- A heuristic is a way which might not always be guaranteed for best solutions but guaranteed to find a good solution in reasonable time.
- Informed search can solve much complex problem which could not be solved in another way.
- The informed search algorithm is more useful for large search space. Informed search algorithm uses the idea of heuristics, so it is also called Heuristic search.
- So Information about cost to goal taken into account
- Ex: Best first search , A* search

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

- It takes the current state of the agent as its input and produces the estimation of how close agent is from the goal.
- The heuristic method, however, might not always give the best solution, but it guaranteed to find a good solution in reasonable time.
- Heuristic function estimates how close a state is to the goal. It is represented by h(n), and it calculates the cost of an optimal path between the pair of states.
- The value of the heuristic function is always positive, it is also called Heuristic search.
- Admissibility of the heuristic function is given as:

  $$h(n) <= h^*(n)$$

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

Pure Heuristic Search

- Pure heuristic search is the simplest form of heuristic search algorithms. It expands nodes based on their heuristic value h(n). It maintains two lists, OPEN and CLOSED list. In the CLOSED list, it places those nodes which have already expanded and in the OPEN list, it places nodes which have yet not been expanded.

- On each iteration, each node n with the lowest heuristic value is expanded and generates all its successors and n is placed to the closed list. The algorithm continues unit a goal state is found.
- In the informed search we will discuss two main algorithms which are given below:

  Best First Search Algorithm (Greedy search)
  A* Search Algorithm

Best-first Search Algorithm (Greedy Search)

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

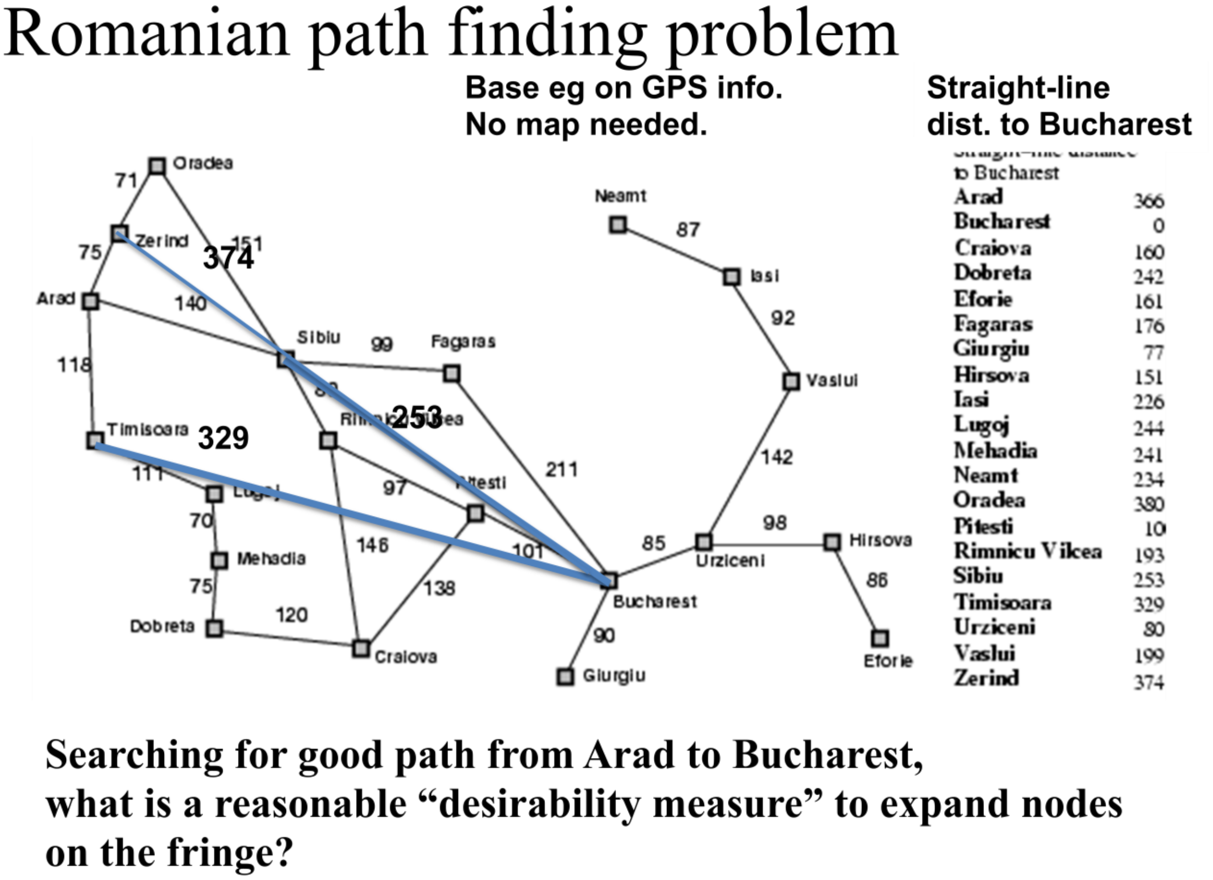
  i.e. $f(n)= g(n)$.

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

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

Example:

- Let us see how this works for route-finding problems in Romania; we use the **straight line distance** heuristic, which we will call h_{SLD}.
- If the goal is Bucharest, we need to know the straight-line distances to Bucharest, which are shown in below figure.
- For example, h_{SLD}(In(Arad))=366. Notice that the values of h_{SLD} cannot be computed from the problem description itself.
- Moreover, it takes a certain amount of experience to know that h_{SLD} is correlated with actual road distances and is, therefore, a useful heuristic.



Romanian path finding problem

Base eg on GPS info.
No map needed.

Straight-line dist. to Bucharest

Straight–line distance to Bucharest	
Arad	366
Bucharest	0
Craiova	160
Dobreta	242
Eforie	161
Fagaras	176
Giurgiu	77
Hirsova	151
Iasi	226
Lugoj	244
Mehadia	241
Neamt	234
Oradea	380
Pitesti	10
Rimnicu Vilcea	193
Sibiu	253
Timisoara	329
Urziceni	80
Vaslui	199
Zerind	374

Searching for good path from Arad to Bucharest, what is a reasonable “desirability measure” to expand nodes on the fringe?

- The above shows the progress of a greedy best-first search using h_{SLD} to find a path from Arad to Bucharest.
- The first node to be expanded from Arad will be Sibiu because it is closer to Bucharest than either Zerind or Timisoara.
- The next node to be expanded will be Fagaras because it is closest. Fagaras in turn generates Bucharest, which is the goal.
- 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. It is not

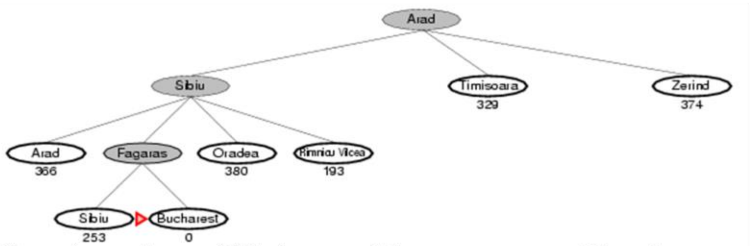
optimal, however: the path via Sibiu and Fagaras to Bucharest is 32 kilometers longer than the path through Rimnicu Vilcea and Pitesti.

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

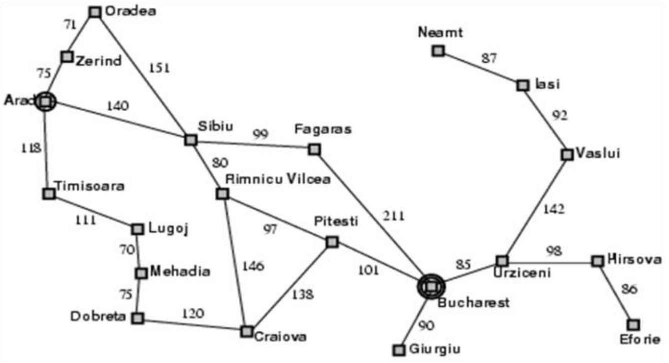
Greedy best-first search example

Straight-line dist. to Bucharest

Arad	366
Bucharest	0
Craiova	160
Dobreta	242
Eforie	161
Fagaras	176
Giurgiu	77
Hirsova	151
Iasi	226
Lugoj	244
Mehadia	241
Neamt	234
Oradea	380
Pitesti	10
Rimnicu Vilcea	193
Sibiu	253
Timisoara	329
Urziceni	80
Vaslui	199
Zerind	374



So, Arad --- Sibiu --- Fagaras --- Bucharest
140+99+211 = 450

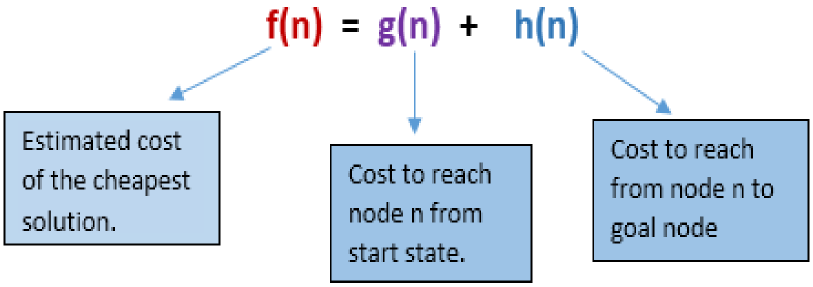


Properties of greedy best-first search:

- Greedy best-first tree search is also incomplete even in a finite state space, much like depth-first search.
- Consider the problem of getting from Iasi to Fagaras. The heuristic suggests that Neamt be expanded first because it is closest to Fagaras, but it is a dead end.
- The solution is to go first to Vaslui—a step that is actually farther from the goal according to the heuristic—and then to continue to Urziceni, Bucharest, and Fagaras.
- The algorithm will never find this solution, however, because expanding Neamt puts Iasi back into the frontier,
- Iasi is closer to Fagaras than Vaslui is, and so Iasi will be expanded again, leading to an infinite loop.
- The worst-case time and space complexity for the tree version is $O(b^m)$,
  where b is the maximum branching factor of the search tree
  m is the maximum depth of the search space.
- With a good heuristic function, however, the complexity can be reduced substantially.
- The amount of the reduction depends on the particular problem and on the quality of the heuristic.

A* search

- A* search is the most commonly known form of best-first search. It uses heuristic function h(n), and cost to reach the node n from the start state g(n).
- A* search algorithm finds the shortest path through the search space using the heuristic function.
- This search algorithm expands less search tree and provides optimal result faster.
- In A* search algorithm, we use search heuristic as well as the cost to reach the node. Hence we can combine both costs as following, and this sum is called as a **fitness number**.



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

Estimated cost of the cheapest solution.

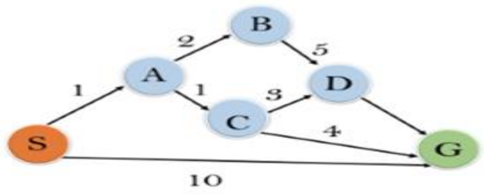
Cost to reach node n from start state.

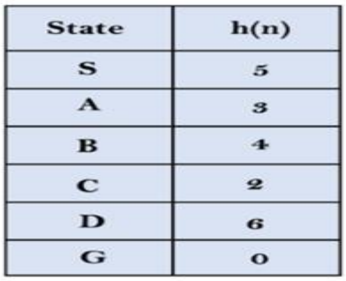
Cost to reach from node n to goal node

Algorithm A*:

```
OPEN = nodes on frontier            CLOSED=expanded nodes
OPEN = {<s,nil>}
while OPEN is not empty
    remove from OPEN the node <n,p> with minimum f(n)
    place <n,p> on CLOSED
    if n is a goal node, return success (path p)
    for each edge connecting n & m with cost c
        if <m,q> is on CLOSED and {p|e} is cheaper than q
            then remove n from CLOSED , put <m,{p|e}> on OPEN
        else if <m,q> is on OPEN AND {p|e} is cheaper than q
            then replace q with {p|e}
        else if  m is not on OPEN put <m,{p|e}> on OPEN
return failure
```

Example 1:

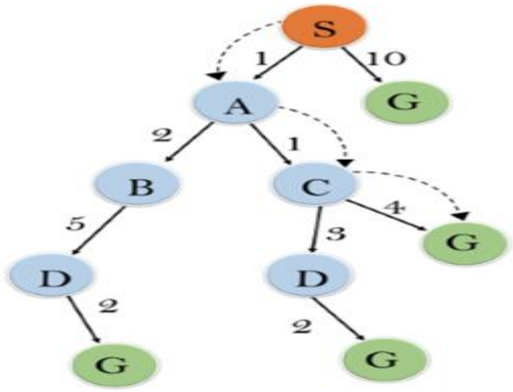




State	h(n)
S	5
A	3
B	4
C	2
D	6
G	0

:

Solution:

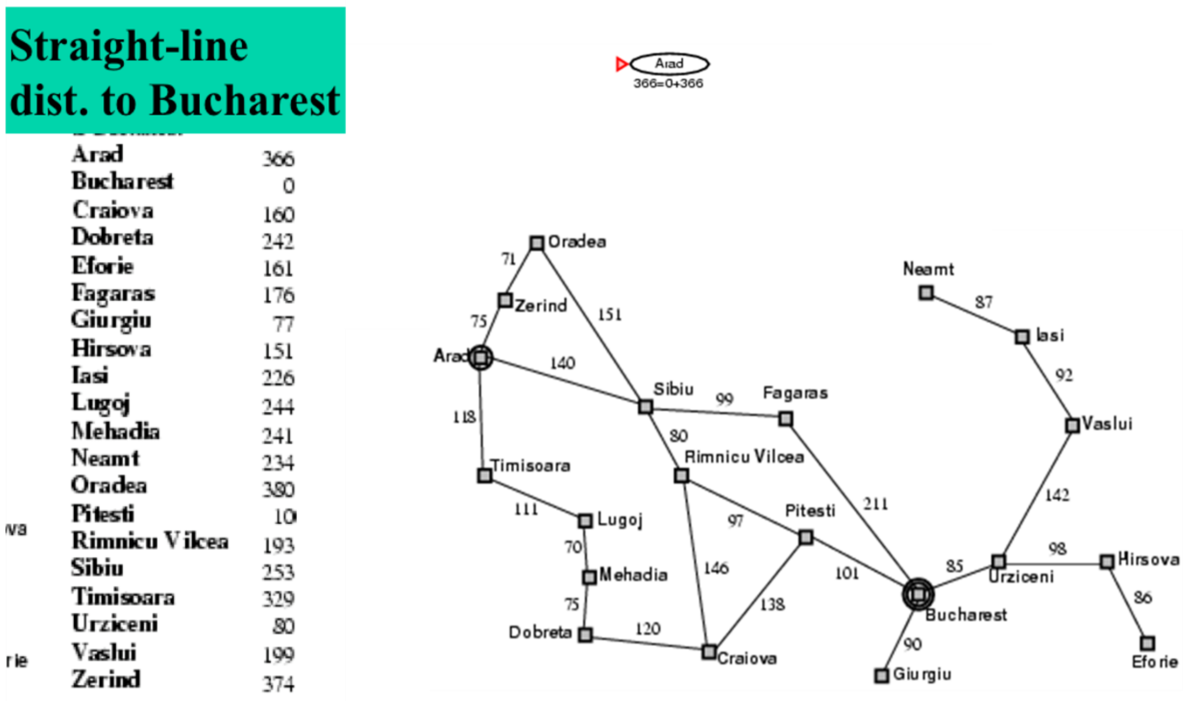


Start State: {(S, 5)}
Iteration1: {(S--> A, 4), (S-->G, 10)}
Iteration2: {(S--> A-->C, 4), (S--> A-->B, 7), (S-->G, 10)}
Iteration 3: {(S--> A-->C--->G, 6), (S--> A-->C--->D, 11), (S--> A-->B, 7), (S-->G, 10)}
Iteration 4 will give the final result, as **S--->A--->C--->G** it provides the optimal path with cost 6

So A* avoid expanding paths that are already expensive

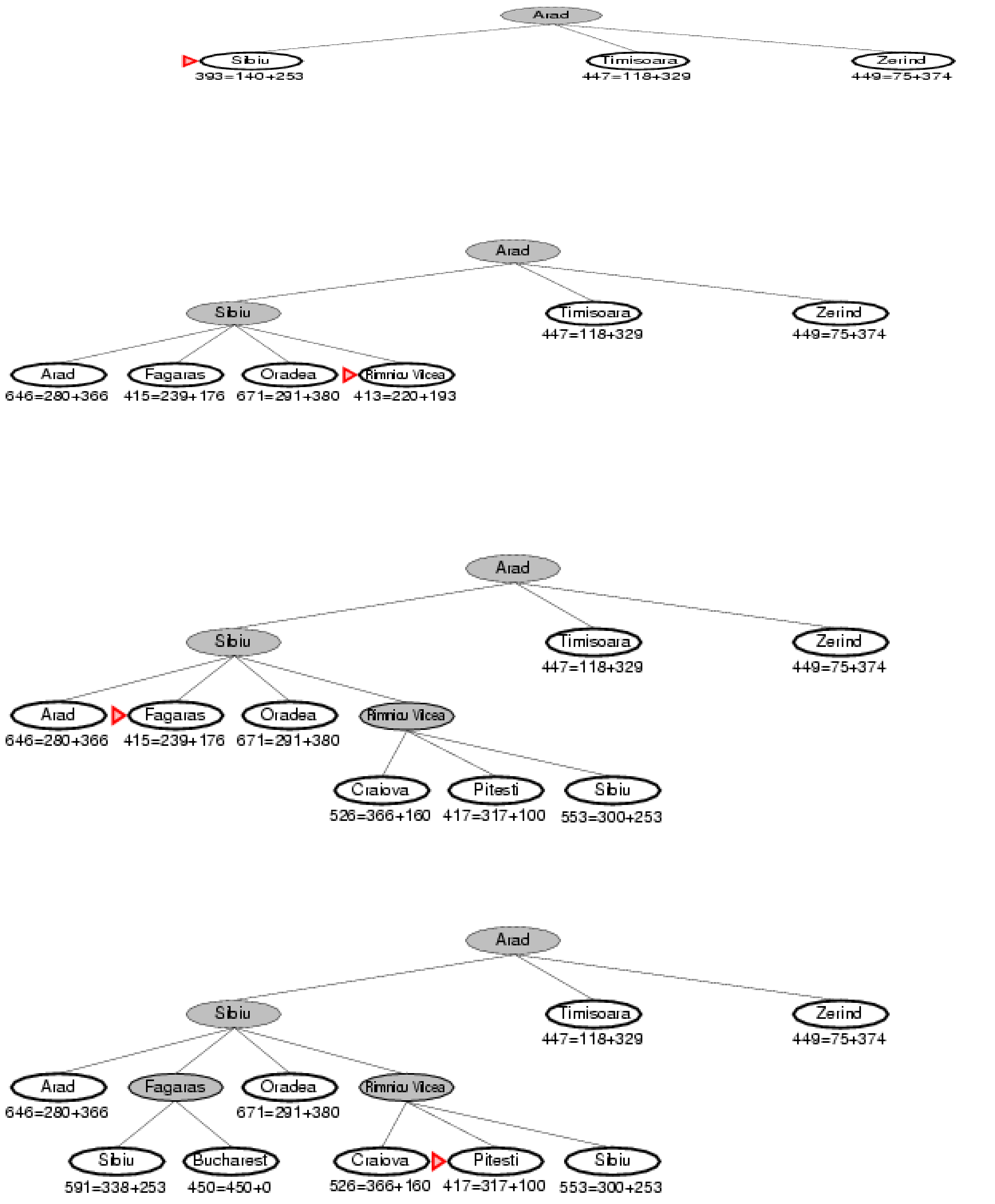
Using: f(n) = g(n) + h(n)

A* search example

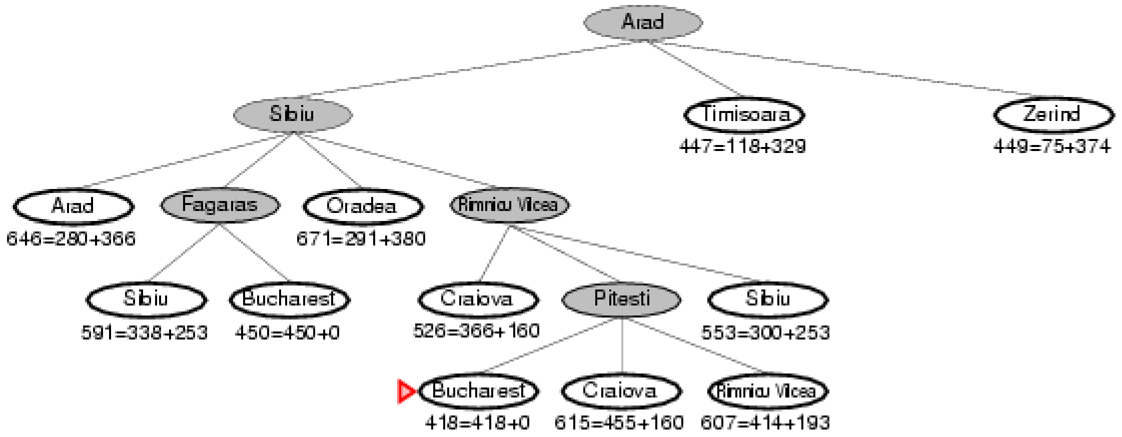


Straight-line dist. to Bucharest

Arad	366
Bucharest	0
Craiova	160
Dobreta	242
Eforie	161
Fagaras	176
Giurgiu	77
Hirsova	151
Iasi	226
Lugoj	244
Mehadia	241
Neamt	234
Oradea	380
Pitesti	10
Rimnicu Vilcea	193
Sibiu	253
Timisoara	329
Urziceni	80
Vaslui	199
Zerind	374



Bucharest appears on the fringe but not selected for expansion since its cost (450) is higher than that of Pitesti (417). Important to understand for the proof of optimality of A*



Optimal Path Found: Arad --- Sibiu --- Rimnicu --- Pitesti --- Bucharest

Under some reasonable conditions for the heuristics, we have:

- **Complete :** Yes, unless there are infinitely many nodes with f(n) < f(Goal)
- **Time Complexity:** The time complexity of A* search algorithm depends on heuristic function, and the number of nodes expanded is exponential to the depth of solution d. So the time complexity is O(b^d) , where b is the branching factor.
- **Space Complexity:** The space complexity of A* search algorithm is O(b^d)
- **Optimal:** Yes

  Also, optimal use of heuristics information!
- Widely used. E.g. Google maps.
- After almost 40 yrs, still new applications found.
- Also, optimal use of heuristic information.

A*: Difficulties

- It becomes often difficult to use A* as the OPEN queue grows very large.
- A solution is to use algorithms that work with less memory
- Memory bounded heuristic search algorithms reduce the memory requirements for A* by introducing IDA*.
- IDA* is an iterative deepening algorithm.
- The cut-off for nodes expanded in an iteration is decided by the f-value of the nodes.

IDA* Algorithm

- IDA* = A* + Iterative Deepening DFS
- The key feature of the IDA* algorithm is that it doesn’t keep a track of each visited node which helps in saving memory consumption and can be used where memory is constrained.
- It is path search algorithm which finds shortest path from start node to goal node in weighted graph.
- It borrows the idea to use a heuristic function from A*

- Its memory usage is weaker than in A*, thereby reducing memory problem in A*
- IDA* is optimal in terms of space and cost
- Problem: Excessive node generation

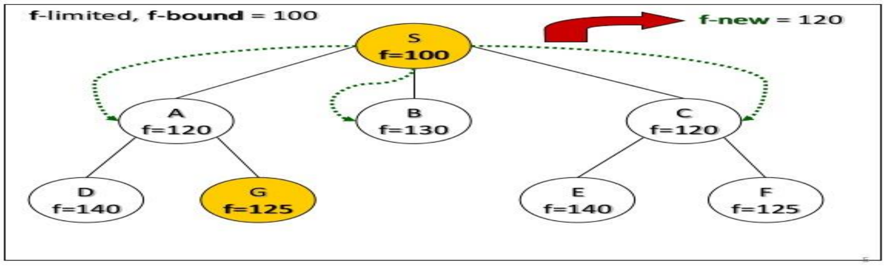
Algorithm:

- Set certain threshold/f-bound
- If f(node) > threshold/f-bound, prune the node
- Set threshold/f-bound = minimum cost of any node that is pruned
- Terminates when goal is reached.

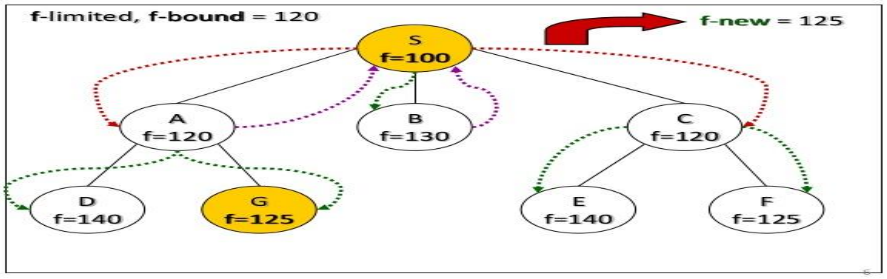
IDA* Search Procedure

- Consider the threshold/f-bound value and cycle through the algorithm
- In every branch visit the depth till the f(node) is greater than the threshold/f-bound and note down that f(node) value, do this till all branches are explored upto certain depth.
- Then the cycle continues from the starting node again with the new threshold/f-new value that is the minimum of f(node) values noted down.
- This continues until the goal is found or the time limit is exceeded

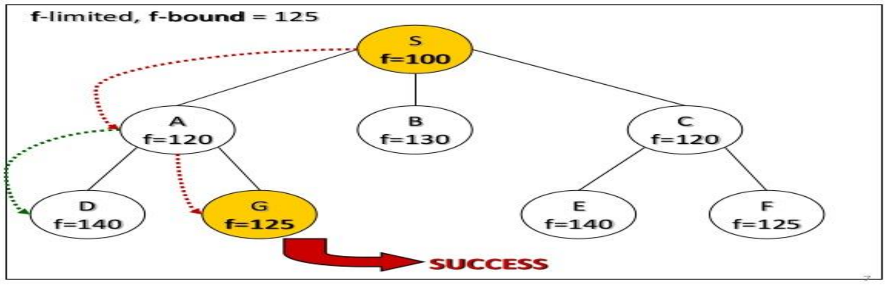
Example



Example



Example



Properties of IDA*:

- **Complete:** Yes, similar to A*
- **Time**: Depends strongly on the number of different values that the heuristic value can take on. If A* expands N nodes, IDA* expands 1+2+……+N=O(N^2) nodes.
- **Space:** It is DFS, it only requires space proportional to the longest path it explores.
- **Optimal:** Yes, similar to A*.

1.4 Adversarial Search

Introduction:

- Adversarial search is a **game-playing** technique where the agents are surrounded by a competitive environment.
- A conflicting goal is given to the agents (multi agent). These agents compete with one another and try to defeat one another in order to win the game. Such conflicting goals give rise to the adversarial search.
- Here, game-playing means discussing those games where **human intelligence** and **logic factor** is used, excluding other factors such as **luck factor**.
- **Tic-tac-toe, chess, checkers**, etc., are such type of games where no luck factor works, only mind works.
- Mathematically, this search is based on the concept of **'Game Theory.'**
- According to game theory, a game is played between two players. To complete the game, one has to win the game and the other looses automatically.

Perfect decision games:

- In game theory, a sequential game has **perfect information** if each player, when making any decision, is perfectly informed of all the events that have previously occurred, including the "initialization event" of the game (e.g. the starting hands of each player in a card game).
- Perfect information is importantly different from complete information, which implies common knowledge of each player's utility functions, payoffs, strategies and "types". A game with perfect information may or may not have complete information.
- Chess is an example of a game with perfect information as each player can see all the pieces on the board at all times.
- Other examples of games with perfect information include tic-tac-toe, checkers, infinite chess.

Imperfect decision games:

- In game theory, a game is imperfect game if each player, when making any decision, is uninformed of all the events that have previously occurred.
- Imperfect information implies no idea of the move taken by the opponent it means the player is unaware of the actions chosen by other players.
- Card games where each player's cards are hidden from other players such as poker and bridge are examples of games with imperfect information

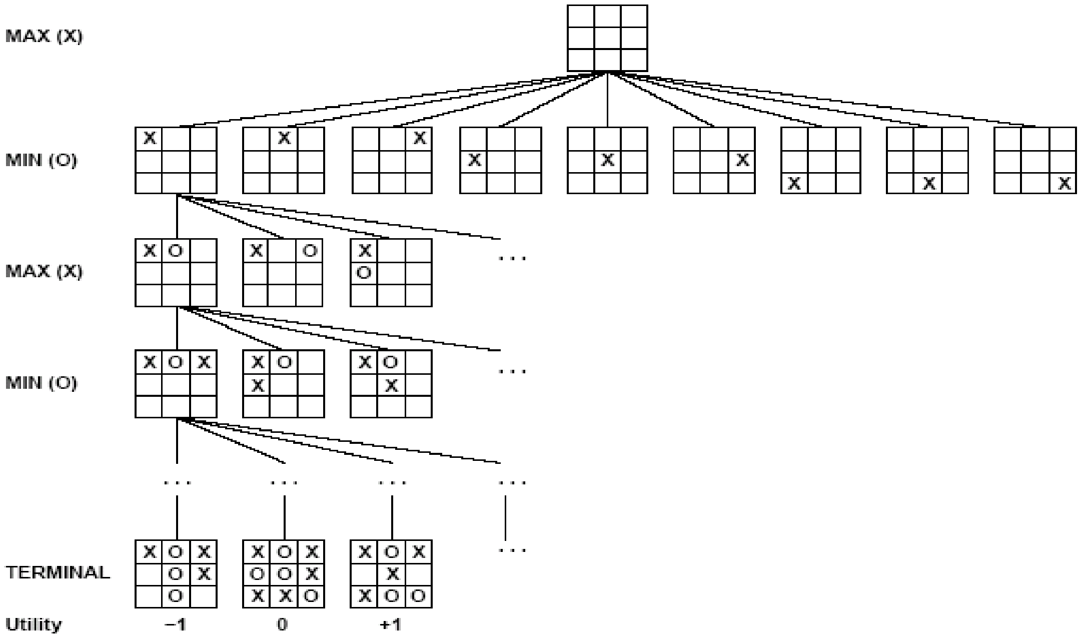
Game Setup:

- Two players: MAX and MIN
- MAX moves first and they take turns until the game is over

  - Winner gets award, loser gets penalty.
- Games as search:
  - Initial state: e.g. board configuration of chess
  - Successor function: list of (move , state) pairs specifying legal moves.
  - Terminal test: Is the game finished?
  - Utility function: Gives numerical value of terminal states.
    E.g. win (+1), lose (-1) and draw (0) in tic-tac-toe or chess

- Size of Search tree is $O(b^d)$
  where b = branching factor , d = number of moves by both players
- Example: Chess :b ~ 35 and D ~100
- So size of search tree is ~ 10^{154} (!!) which completely impractical to search this.
- Game-playing emphasizes being able to make optimal decisions in a finite amount of time, somewhat realistic as a model of a real-world agent and even if games themselves are artificial

Partial Game Tree for Tic-Tac-Toe

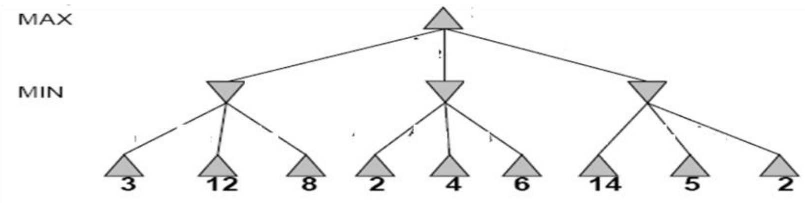


MINI - MAX ALGORITHM:

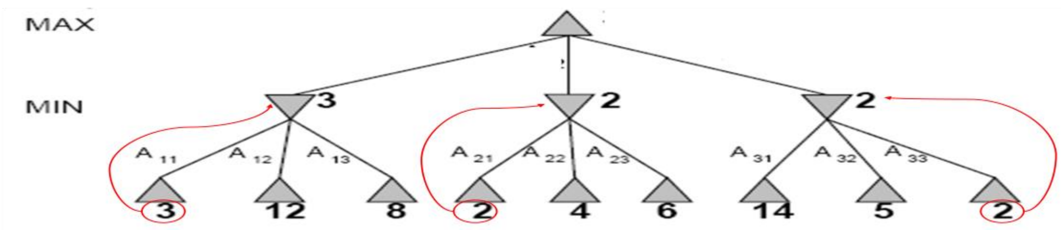
- Mini-max algorithm is a recursive or backtracking algorithm which is used in decision-making and game theory. It provides an optimal move for the player assuming that opponent is also playing optimally.
- Mini-Max algorithm uses recursion to search through the game-tree.

- Min-Max algorithm is mostly used for game playing in AI. Such as Chess, Checkers, tic-tac-toe, go, and various tow-players game. This Algorithm computes the minimax decision for the current state.
- In this algorithm two players play the game, one is called MAX and other is called MIN.
- Both the players fight it as the opponent player gets the minimum benefit while they get the maximum benefit.
- Both Players of the game are opponent of each other, where MAX will select the maximized value and MIN will select the minimized value.
- The minimax algorithm performs a depth-first search algorithm for the exploration of the complete game tree.
- The minimax algorithm proceeds all the way down to the terminal node of the tree, then backtrack the tree as the recursion.

Two-Ply Game Tree

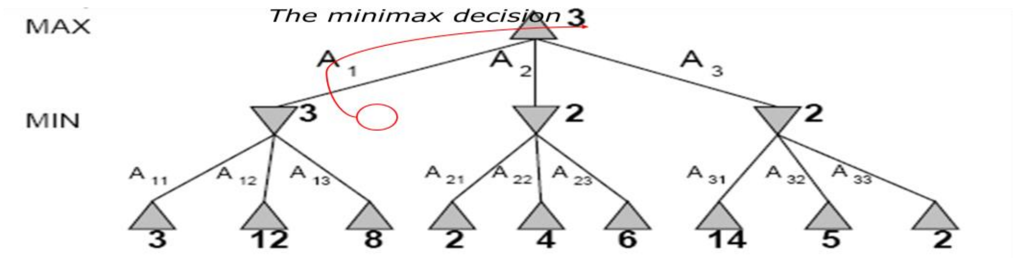


Two-Ply Game Tree



Two-Ply Game Tree

Minimax maximizes the utility for the worst-case outcome for max



Pseudo code for Minimax Algorithm:

```
function MINIMAX-DECISION(state) returns an action
inputs: state, current state in game
v←MAX-VALUE(state)
  return the action in SUCCESSORS(state) with value v

function MAX-VALUE(state) returns a utility value
  if TERMINAL-TEST(state) then return UTILITY(state)
v ← -∞
  for a,s in SUCCESSORS(state) do
v ←MAX(v,MIN-VALUE(s))
  return v

function MIN-VALUE(state) returns a utility value
  if TERMINAL-TEST(state) then return UTILITY(state)
v ← -∞
  for a,s in SUCCESSORS(state) do
v ←MIN(v,MAX-VALUE(s))
  return v
```

Draw back:

- The main drawback of the minimax algorithm is that it gets really slow for complex games such as Chess, go, etc.
- This type of games has a huge branching factor, and the player has lots of choices to decide.
- This limitation of the minimax algorithm can be improved from **alpha-beta pruning**

Static (Heuristic) Evaluation Functions:

An **Evaluation Function:**

- estimates how good the current board configuration is for a player.

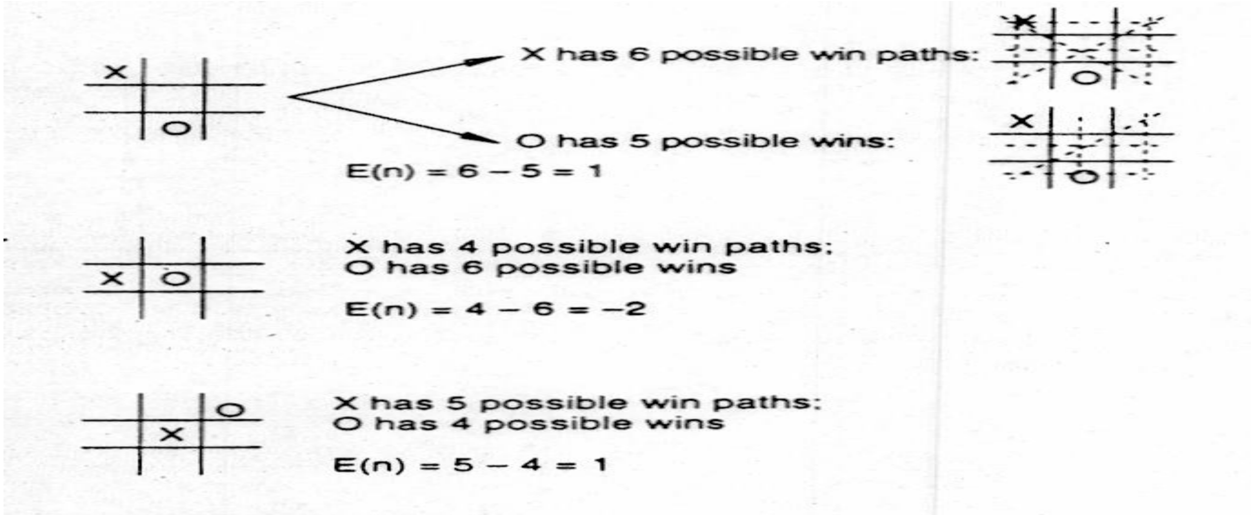
- Typically, one figures how good it is for the player, and how good it is for the opponent, and subtracts the opponents score from the players
- Tic-tac-toe : Number of successful moves by X – Number of successful moves by O
- Chess: Value of all white pieces - Value of all black pieces
- Typical values from -infinity (loss) to +infinity (win) or [-1, +1].
- If the board evaluation is X for a player, it's -X for the opponent.
- Many clever ideas about how to use the evaluation function.

  e.g. null move heuristic: let opponent move twice.
- Example:

  Evaluating chess boards,

  Tic-tac-toe

The static heuristic evaluation function of Tic-tac-toe



Heuristic is $E(n) = M(n) - O(n)$

where $M(n)$ is the total of My possible winning lines

$O(n)$ is total of Opponent's possible winning lines

$E(n)$ is the total Evaluation for state n

Figure 4.16 Heuristic measuring conflict applied to states of tic-tac-toe.

α-β Pruning

- Alpha-beta pruning is a modified version of the minimax algorithm. It is an optimization technique for the minimax algorithm.
- As we have seen in the minimax search algorithm that the number of game states it has to examine are exponential in depth of the tree.
- Since we cannot eliminate the exponent, but we can cut it to half. Hence there is a technique by which without checking each node of the game tree we can compute the correct minimax decision, and this technique is called **pruning**.

- This involves two threshold parameter Alpha and beta for future expansion, so it is called **alpha-beta pruning**. It is also called as **Alpha-Beta Algorithm**.
- Alpha-beta pruning can be applied at any depth of a tree, and sometimes it not only prune the tree leaves but also entire sub-tree.

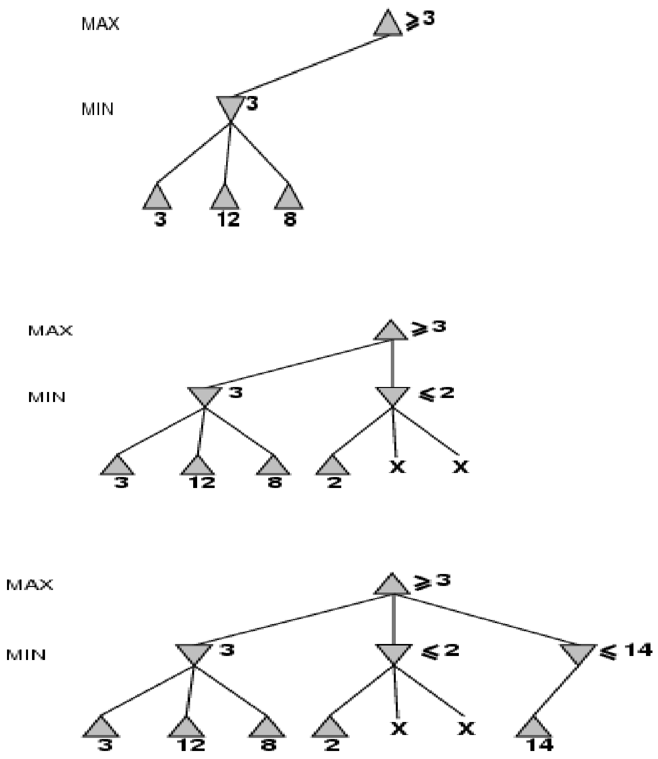
The two-parameter can be defined as:

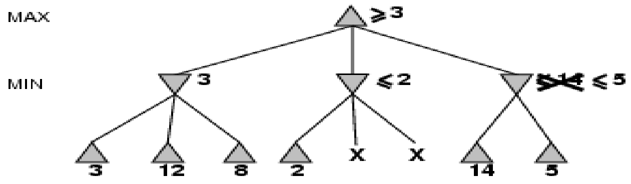
- **Alpha:** The best (highest-value) choice we have found so far at any point along the path of Maximizer. The initial value of alpha is -∞.
- **Beta:** The best (lowest-value) choice we have found so far at any point along the path of Minimizer. The initial value of beta is +∞.
- The Alpha-beta pruning to a standard minimax algorithm returns the same move as the standard algorithm does, but it removes all the nodes which are not really affecting the final decision but making algorithm slow. Hence by pruning these nodes, it makes the algorithm fast.
- Using α-β Pruning improve search by reducing the size of the game tree.

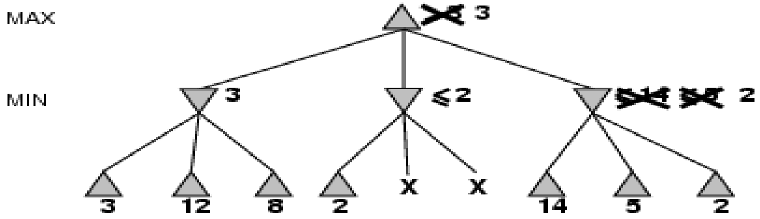
Principle of α-β Pruning

- If a move is determined worse than another move already examined, then there is no need for further examination of the node.

Example 1:



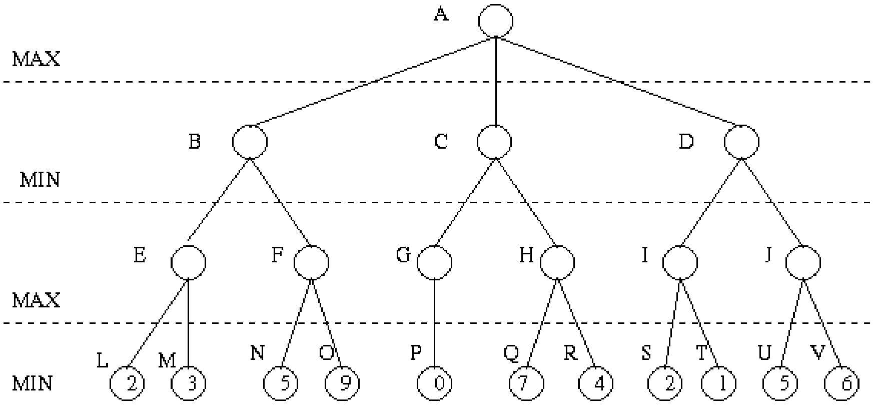


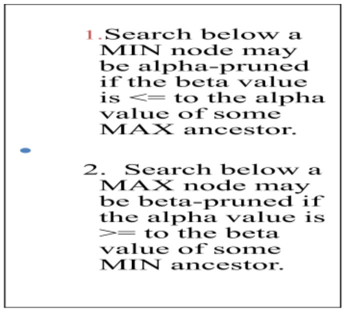


Rules of Thumb

- α is the best (highest)   found so far along the path for Max
- β is the best (lowest) found so far along the path for Min
- Search below a MIN node may be **alpha-pruned** if the its **β ≤ α** of some MAX ancestor
- Search below a MAX node may be **beta-pruned** if the its **α≥ β** of some MIN ancestor.

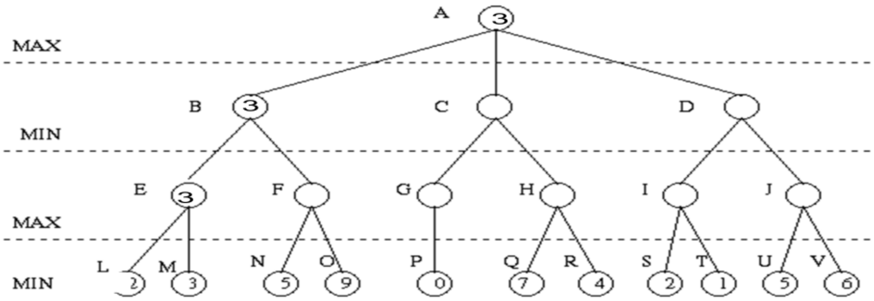
Example 2:





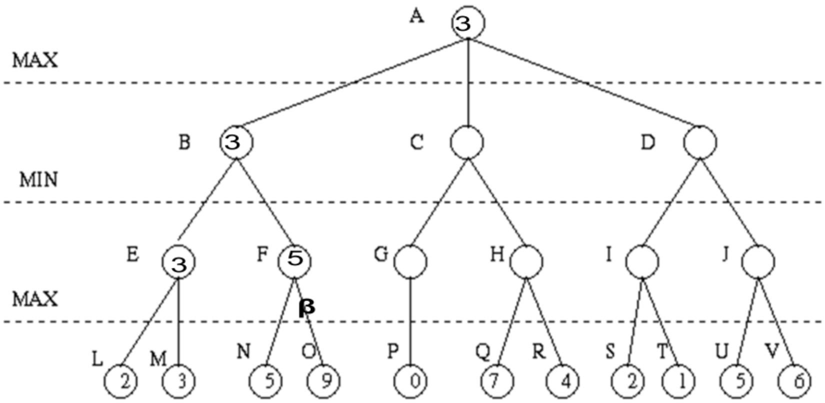
1. Search below a MIN node may be alpha-pruned if the beta value is <= to the alpha value of some MAX ancestor.

2. Search below a MAX node may be beta-pruned if the alpha value is >= to the beta value of some MIN ancestor.



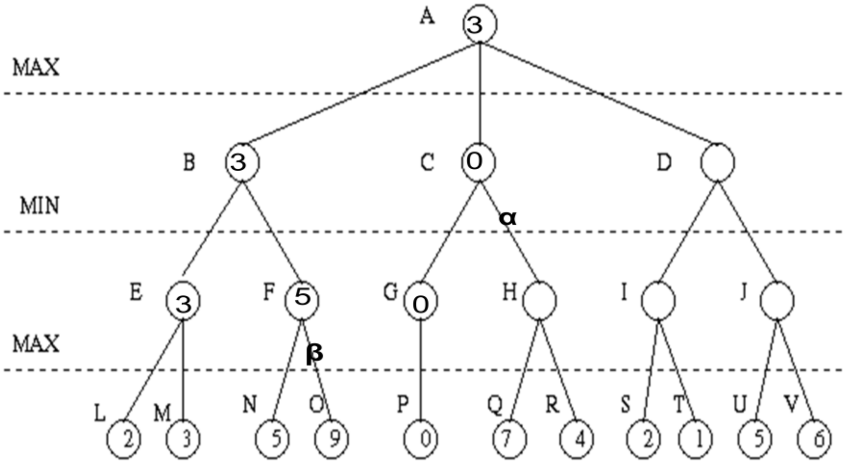
1. Search below a MIN node may be alpha-pruned if the beta value is <= to the alpha value of some MAX ancestor.

2. Search below a MAX node may be beta-pruned if the alpha value is >= to the beta value of some MIN ancestor.



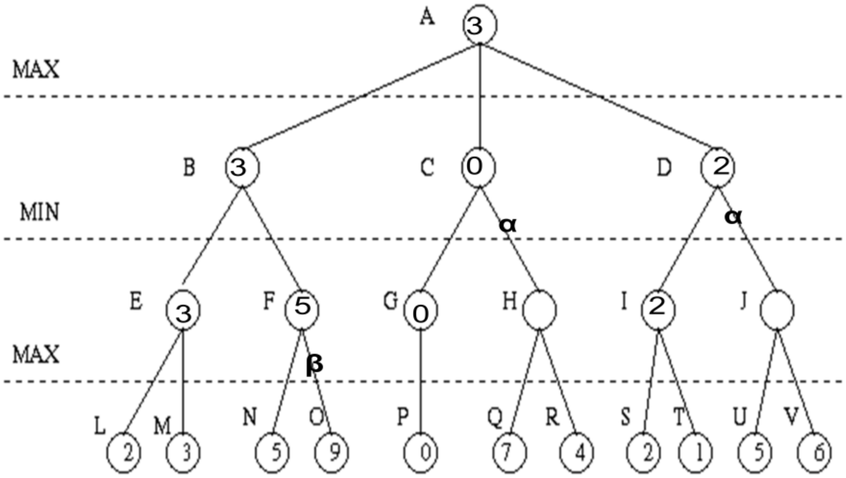
1. Search below a MIN node may be alpha-pruned if the beta value is <= to the alpha value of some MAX ancestor.

2. Search below a MAX node may be beta-pruned if the alpha value is >= to the beta value of some MIN ancestor.

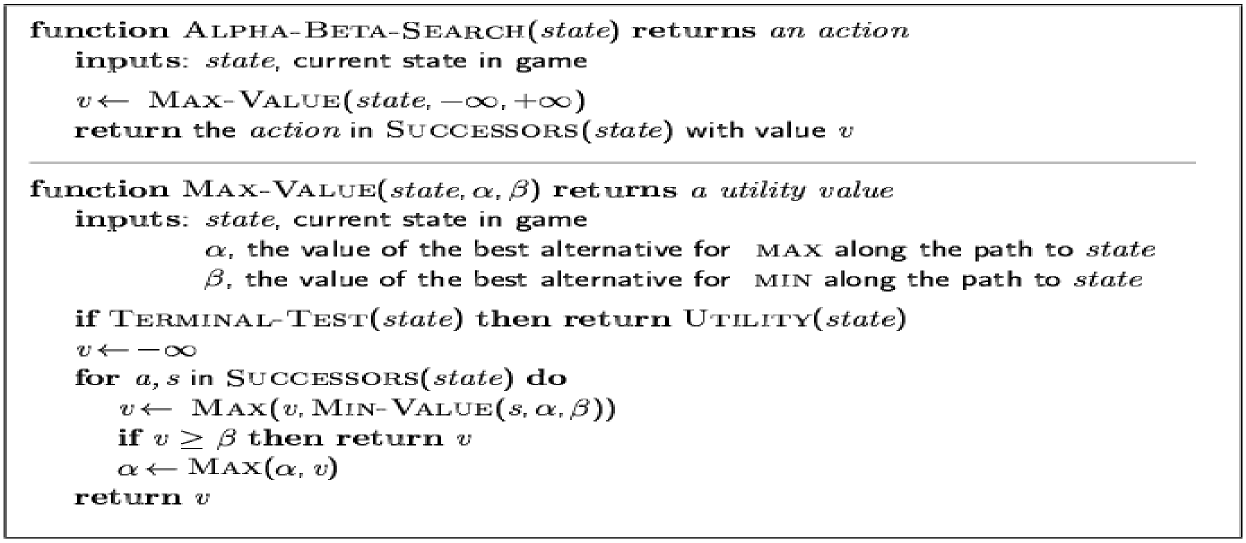


1. Search below a MIN node may be alpha-pruned if the beta value is <= to the alpha value of some MAX ancestor.

2. Search below a MAX node may be beta-pruned if the alpha value is >= to the beta value of some MIN ancestor.

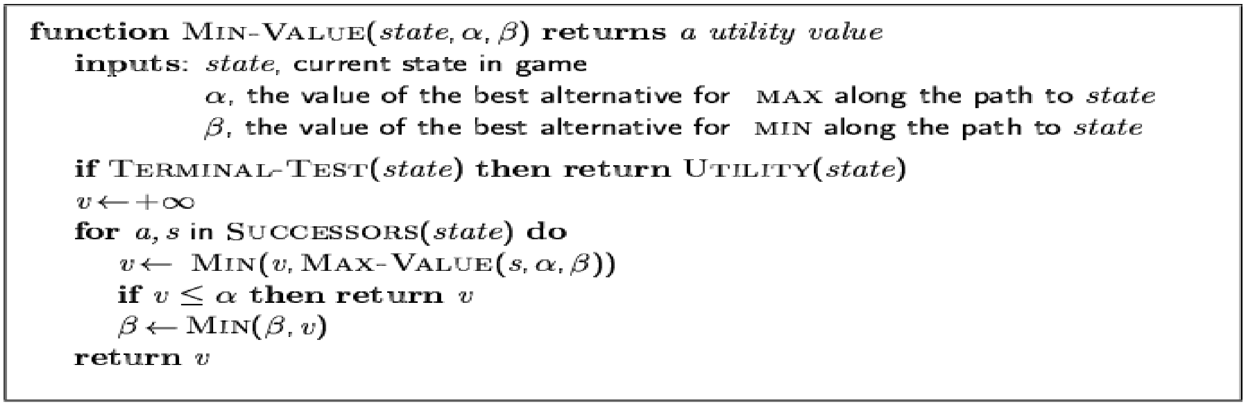


The α-β algorithm



```
function ALPHA-BETA-SEARCH(state) returns an action
   inputs: state, current state in game
   $v \leftarrow$ MAX-VALUE(state, $-\infty, +\infty$)
   return the action in SUCCESSORS(state) with value $v$

function MAX-VALUE(state, $\alpha$, $\beta$) returns a utility value
   inputs: state, current state in game
           $\alpha$, the value of the best alternative for MAX along the path to state
           $\beta$, the value of the best alternative for MIN along the path to state
   if TERMINAL-TEST(state) then return UTILITY(state)
   $v \leftarrow -\infty$
   for $a, s$ in SUCCESSORS(state) do
      $v \leftarrow$ MAX($v$, MIN-VALUE($s, \alpha, \beta$))
      if $v \geq \beta$ then return $v$
      $\alpha \leftarrow$ MAX($\alpha$, $v$)
   return $v$
```



```
function MIN-VALUE(state, $\alpha$, $\beta$) returns a utility value
   inputs: state, current state in game
           $\alpha$, the value of the best alternative for MAX along the path to state
           $\beta$, the value of the best alternative for MIN along the path to state
   if TERMINAL-TEST(state) then return UTILITY(state)
   $v \leftarrow +\infty$
   for $a, s$ in SUCCESSORS(state) do
      $v \leftarrow$ MIN($v$, MAX-VALUE($s, \alpha, \beta$))
      if $v \leq \alpha$ then return $v$
      $\beta \leftarrow$ MIN($\beta$, $v$)
   return $v$
```

Properties of α-β Prune:

- Pruning does not affect final result
- Good move ordering improves effectiveness of pruning b(e.g., chess, try captures first, then threats, forward moves, then backward moves…)
- With "perfect ordering," time complexity = O($b^{m/2}$)
    - → doubles depth of search that alpha-beta pruning can explore

Artificial Intelligence (CSE)
UNIT-II

2.1.1 Logical Agents:

Knowledge-based agents – agents that have an explicit representation of knowledge that can be reasoned with.

These agents can manipulate this knowledge to infer new things at the “knowledge level”

A Knowledge Based Agent

- A knowledge-based agent includes a knowledge base and an inference system.
- A knowledge base is a set of representations of facts of the world.
- Each individual representation is called a sentence.
- The sentences are expressed in a knowledge representation language.
- The agent operates as follows:
    1. It TELLs the knowledge base what it perceives.
    2. It ASKs the knowledge base what action it should perform.
    3. It performs the chosen action.

Architecture of Knowledge based agent:

Knowledge Level.

- The most abstract level: describe agent by saying what it knows.
- Example: A taxi agent might know that the Golden Gate Bridge connects San Francisco with the Marin County.

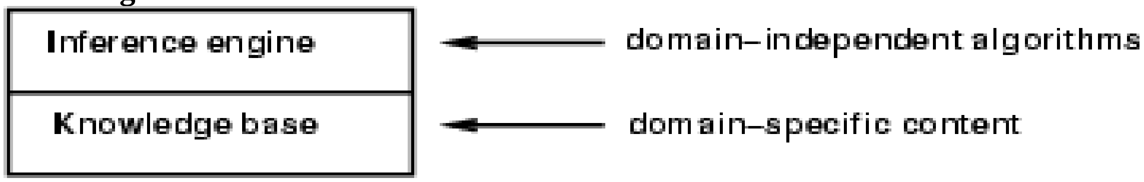
Logical Level.

- The level at which the knowledge is encoded into sentences.
- Example: Links(GoldenGateBridge, SanFrancisco, MarinCounty).

Implementation Level.

- The physical representation of the sentences in the logical level.
- Example: ‘(links goldengatebridge sanfrancisco marincounty)

Knowledge Bases:

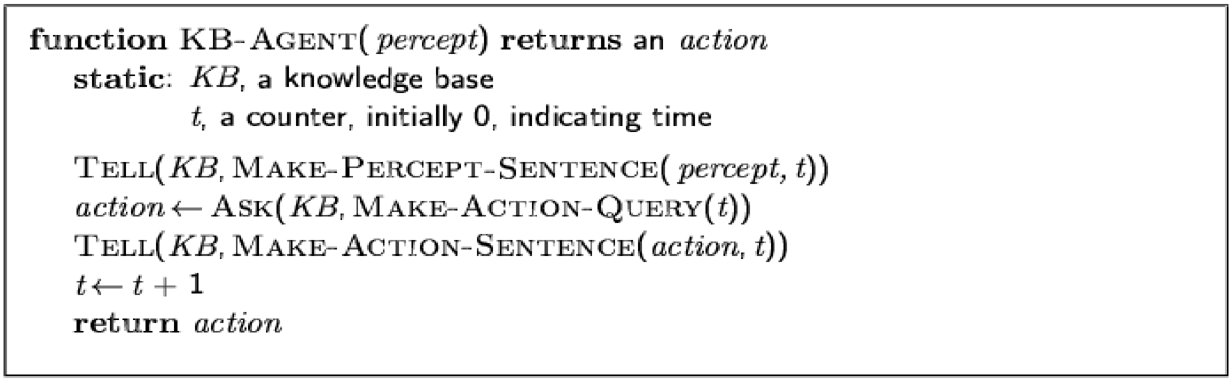


- Knowledge base = set of sentences in a formal language
- Declarative approach to building an agent (or other system):

  Tell it what it needs to know
- Then it can Ask itself what to do - answers should follow from the KB
- Agents can be viewed at the knowledge level - i.e., what they know, regardless of how implemented
- Or at the implementation level

i.e., data structures in KB and algorithms that manipulate them

A Simple Knowledge Based Agent



```
function KB-AGENT(percept) returns an action
    static: KB, a knowledge base
            t, a counter, initially 0, indicating time

    TELL(KB, MAKE-PERCEPT-SENTENCE(percept, t))
    action ← ASK(KB, MAKE-ACTION-QUERY(t))
    TELL(KB, MAKE-ACTION-SENTENCE(action, t))
    t ← t + 1
    return action
```

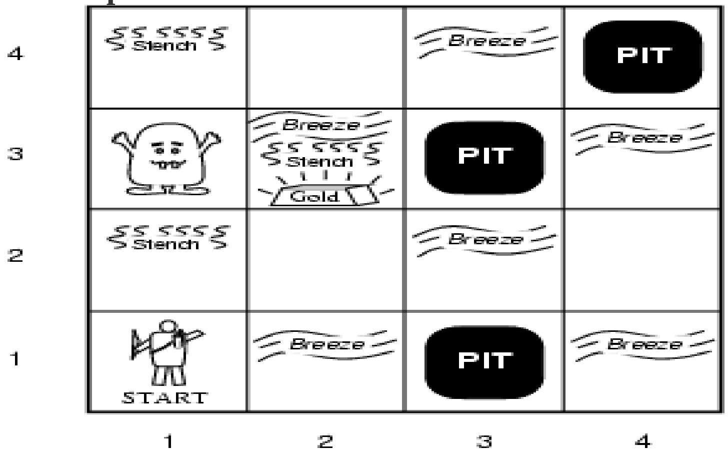
The agent must be able to:

- Represent states, actions, etc.
- Incorporate new percepts
- Update internal representations of the world
- Deduce hidden properties of the world
- Deduce appropriate actions

The Wumpus world environment

- The Wumpus computer game
- The agent explores a cave consisting of rooms connected by passageways.
- Lurking somewhere in the cave is the Wumpus, a beast that eats any agent that enters its room.
- Some rooms contain bottomless pits that trap any agent that wanders into the room.
- Occasionally, there is a heap of gold in a room.
- The goal is to collect the gold and exit the world without being eaten

A Wumpus World



A Wumpus World PEAS description

Performance measure:

- gold +1000, death -1000
- -1 per step, -10 for using the arrow

Environment: 4 x 4 grid of rooms

- Squares adjacent to wumpus are smelly
- Squares adjacent to 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 gold if in same square
- Releasing drops the gold in same square

Actuators: Left turn, Right turn, Forward, Grab, Release, Shoot

Sensors: Stench, Breeze, Glitter, Bump, Scream (shot Wumpus)

Agents in a Wumpus World: Percepts

The agent perceives

- a stench in the square containing the wumpus and in the adjacent squares (not diagonally)
- a breeze in the squares adjacent to a pit
- a glitter in the square where the gold is
- a bump, if it walks into a wall
- a woeful scream everywhere in the cave, if the wumpus is killed

The percepts are given as a five-symbol list. If there is a stench and a breeze, but no glitter, no bump, and no scream, the percept is [Stench, Breeze, None, None, None]

Percepts given to the agent

- Stench
- Breeze
- Glitter
- Bumb (ran into a wall)
- Scream (wumpus has been hit by arrow)

Principle Difficulty: Agent is initially ignorant of the configuration of the environment – going to have to reason to figure out where the gold is without getting killed!

Wumpus World Actions

- **go forward**
- **turn right**
- **turn left**
- **grab**: Pick up an object that is in the same square as the agent

- **shoot**: Fire an arrow in a straight line in the direction the agent is facing. The arrow continues until it either hits and kills the wumpus or hits the outer wall. The agent has only one arrow, so only the first Shoot action has any effect
- **climb** is used to leave the cave. This action is only effective in the start square
- **die**: This action automatically and irretrievably happens if the agent enters a square with a pit or a live wumpus

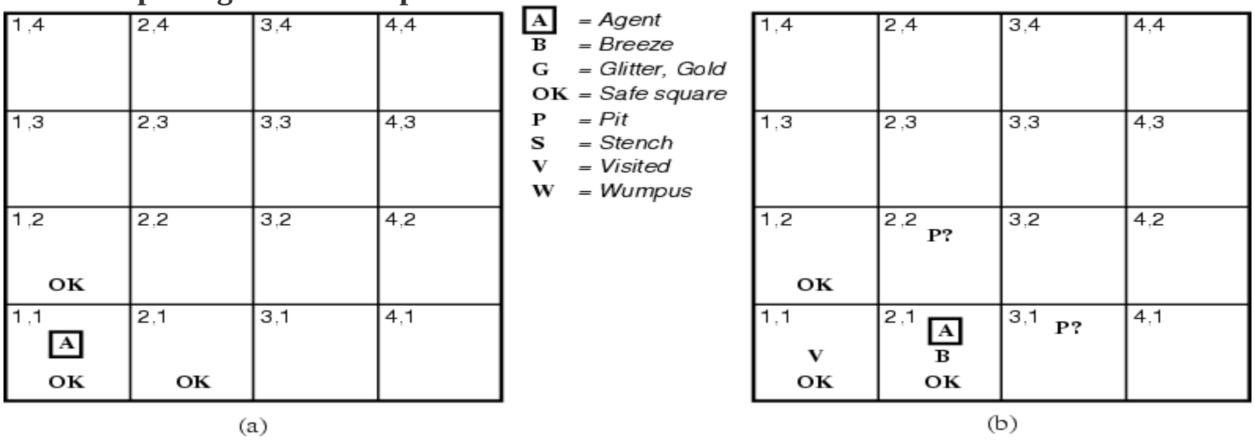
Wumpus goal:

The agent's goal is to find the gold and bring it back to the start square as quickly as possible, without getting killed

- 1000 points reward for climbing out of the cave with the gold
- 1 point deducted for every action taken
- 10000 points penalty for getting killed

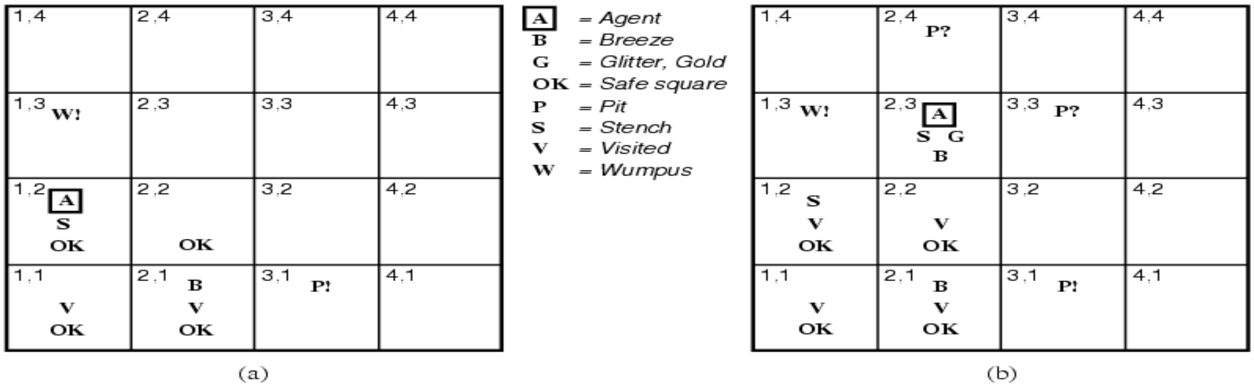
Note that in each case for which the agent draws a conclusion from the available information, that conclusion is *guaranteed to be correct if the available information is correct.* This is a fundamental property of logical reasoning.

The Wumpus Agent First Step



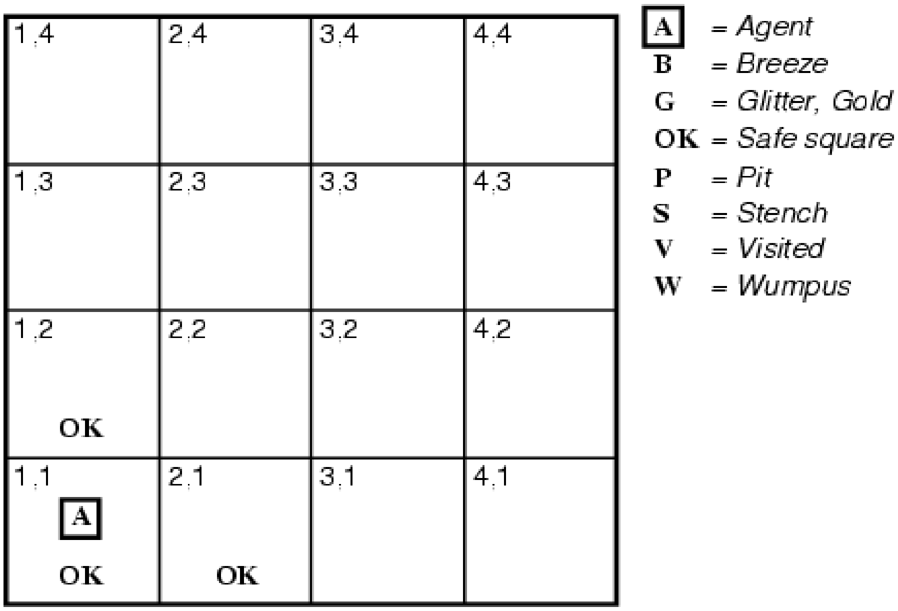
(a) (b)

Later



(a) (b)

Let's Play



Wumpus World Characterization:

- **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

2.1.2 Logic in general:

- Logics are formal languages for representing information such that conclusions can be drawn
- Syntax defines how symbols can be put together to form the sentences in the language
- Semantics define the "meaning" of sentences;
    - i.e., define truth of a sentence in a world (given an interpretation)
    - E.g., the language of arithmetic
    - x+2 ≥ y is a sentence; x2+y > {} is not a sentence
    - x+2 ≥ y is true iff the number x+2 is no less than the number y
    - x+2 ≥ y is true in a world where x = 7, y = 1
    - x+2 ≥ y is false in a world where x = 0, y = 6

Propositional Logic:

Propositional logic (PL) is the simplest form of logic where all the statements are made by propositions. A proposition is a declarative statement which is either true or false. It is a technique of knowledge representation in logical and mathematical form. Propositions can be either true or false, but it cannot be both. Propositional logic is also called Boolean logic as it works on 0 and 1.

Examples:

Today is Tuesday.

The Sun rises from West (False proposition)
2+2= 5(False proposition)
2+3= 5

Syntax of Propositional Logic:

The syntax of propositional logic defines the allowable sentences. The atomic sentences consist of a single proposition symbol. Each such symbol stands for a proposition that can be true or false. We use symbols that start with an uppercase letter and may contain other letters or subscripts.

Example: P, Q, R, W1, 3 and North.

The names are arbitrary but are often chosen to have some mnemonic value—we use W1,3 to stand for the proposition that the wumpus is in [1,3]. There are two proposition symbols with fixed meanings:

True is the always-true proposition and
False is the always-false proposition.

Complex sentences are constructed from simpler sentences, using parentheses and logical connectives.

In propositional logic there are two types of propositions. They are:

1. Atomic Propositions
2. Compound Propositions

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

Ex: The sun rises from west . (False Atomic Propositions)
The sun rises from east (True Atomic Propositions)

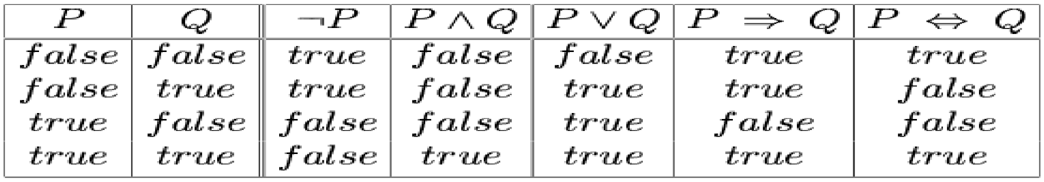
Compound proposition: Compound propositions are constructed by combining simpler or atomic propositions, using parenthesis and logical connectives.

Ex: Arsalan is an engineer and arsalan lives in dubai.
Arsalan is an engineer or arsalan is a doctor.

Propositional logic is the simplest logic – illustrates basic ideas

The proposition symbols P_1, P_2 etc are (atomic) sentences

- If S is a sentence, ¬(S) is a sentence (negation)
- If S_1 and S_2 are sentences, ($S_1 \wedge S_2$) is a sentence (conjunction)
- If S_1 and S_2 are sentences, ($S_1 \vee S_2$) is a sentence (disjunction)
- If S_1 and S_2 are sentences, ($S_1 \Rightarrow S_2$) is a sentence (implication)
- If S_1 and S_2 are sentences, ($S_1 \Leftrightarrow S_2$) is a sentence (biconditional)



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*

Truth Table for connectives

Examples of Connectives:

($P \wedge Q$) Arsalan likes football and Arsalan likes baseball.
($P \vee Q$) Arsalan is a doctor or Arsalan is an engineer.
($P \Rightarrow Q$) If it is raining, then the street is wet.
($P \Leftrightarrow Q$) I am breathing if and only if I am alive

Precedence of Connectives:
To eliminate the ambiguity we define a precedence for each operator. The "not" operator ($\neg$) has the highest precedence, followed by $\wedge$ (conjunction), $\vee$(disjunction),$\Rightarrow$(implication),$\Leftrightarrow$ (biconditional).
Example:$\neg A \wedge B$ the $\neg$ binds most tightly, giving us the equivalent of $(\neg A) \wedge B$ rather than $\neg (A \wedge B)$

Semantics of Propositional Logic:
Each model specifies true/false for each proposition symbol

E.g.	A	B	C
	false	true	false

With these symbols, 8 possible models, can be enumerated automatically.
Rules for evaluating truth with respect to a model *m*:

$\neg S$	is true iff	S is false	
$S_1 \wedge S_2$	is true iff	S_1 is true and	S_2 is true
$S_1 \vee S_2$	is true iff	$S_1$is true or	S_2 is true
$S_1 \Rightarrow S_2$	is true iff	S_1 is false or	S_2 is true
i.e.,	is false iff	S_1 is true and	S_2 is false
$S_1 \Leftrightarrow S_2$	is true iff	$S_1 \Rightarrow S_2$ is true and$S_2 \Rightarrow S_1$ is true	

Simple recursive process evaluates an arbitrary sentence,
e.g., $\neg A \wedge (B \vee C) = \mathit{true} \wedge (\mathit{true} \vee \mathit{false}) = \mathit{true} \wedge \mathit{true} = \mathit{true}$

Limitations of Propositional Logic:
We cannot represent relations like All, some, or none with propositional logic.
Examples:
All boys are smart.
All girls are hardworking.
Some mangoes are sweet.
Propositional logic has limited expressive power and in propositional logic, we cannot describe statements in terms of their properties or logical relationships.

2.1.3 First Order Logic (FOL):

Propositional logic assumes the world contains facts whereas first-order logic (like natural language) assumes the world contains

- **Objects:** people, houses, numbers, colors, baseball games, wars, …
- **Relations**: red, round, prime, brother of, bigger than, part of, comes between, …
- **Functions**: father of, best friend, one more than, plus, …

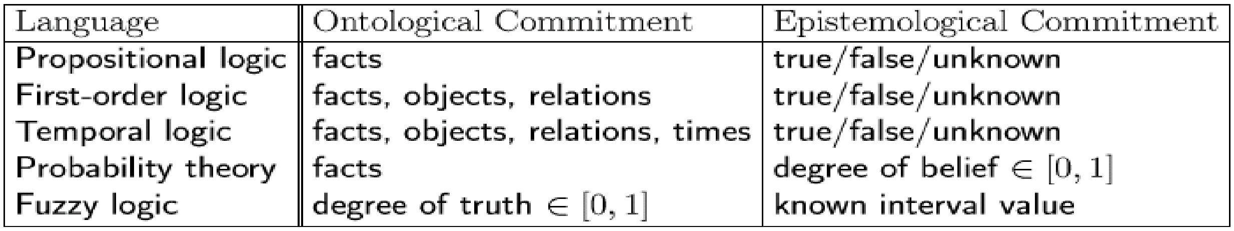
Syntax of FOL:

Constants	KingJohn, 2,...
Predicates	Brother, >,...
Functions	Sqrt, ...
Variables	x, y, a, b,...
Connectives	¬, ⇒, ∧, ∨, ⇔
Equality	=
Quantifiers	∀, ∃

Logics in general:

Ontological Commitment:

- What exists in the world — TRUTH
- PL : facts hold or do not hold.
- FOL : objects with relations between them that hold or do not hold



Language	Ontological Commitment	Epistemological Commitment
Propositional logic	facts	true/false/unknown
First-order logic	facts, objects, relations	true/false/unknown
Temporal logic	facts, objects, relations, times	true/false/unknown
Probability theory	facts	degree of belief $\in [0, 1]$
Fuzzy logic	degree of truth $\in [0, 1]$	known interval value

- Constant Symbols:
  - Stand for objects
  - e.g., KingJohn, 2,...
- Predicate Symbols
  - Stand for relations
  - e.g., Brother(Richard, John), greater_than(3,2)...
- Function Symbols
  - Stand for functions
  - e.g., Sqrt(3) ...

Relations:

- Some relations are properties: they state
- some fact about a single object:
  Round(ball), Prime(7).
- n-ary relations state facts about two or more objects:
  Married(John,Mary), LargerThan(3,2).
- Some relations are functions: their value is another object: Plus(2,3), Father(Dan).

Terms:

- Term = logical expression that refers to an object.
- There are 2 kinds of terms:
  - constant symbols: Table, Computer
  - function symbols: Sqrt(3), Plus(2,3) etc
- Functions can be nested:
  - Pat_Grandfather(x) = father(father(x))
- Terms can contain variables.

- No variables = **ground term**.

Atomic Sentence:

Atomic sentences are the most basic sentences of first-order logic. These sentences are formed from a predicate symbol followed by a parenthesis with a sequence of terms.

Atomic sentence = *predicate* (*term*$_1$,...,*term*$_n$)

Atomic sentences state facts using terms and predicate symbols P(x,y) interpreted as “x is P of y”

Examples: LargerThan(2,3) is false.
Married(Father(Richard), Mother(John)) could be true or false

Complex Sentence:

Complex sentences are made from atomic sentences using connectives

$$\neg S,\ S_1 \wedge S_2,\ S_1 \vee S_2,\ S_1 \Rightarrow S_2,\ S_1 \Leftrightarrow S_2,$$

E.g. *Sibling(KingJohn,Richard)* $\Rightarrow$ *Sibling(Richard,KingJohn)*

Complex sentences are made by combining atomic sentences using connectives.We make complex sentences with connectives (just like in propositional logic).

Some more examples:

1. Brother(Richard, John) $\wedge$ Brother(John, Richard)
2. King(Richard) $\vee$ King(John)
3. King(John) => $\neg$ King(Richard)

Note: Semantics are the same as in propositional logic

Variables:

- Person(John) is true or false because we give it a single argument ‘John’
- We can be much more flexible if we allow variables which can take on values in a domain. e.g., all persons x, all integers i, etc.
  - E.g., can state rules like Person(x) => HasHead(x)
  - or Integer(i) => Integer(plus(i,1))

Quantifiers in First Order Logic:

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

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

Universal Quantifier:

Universal quantifier is a symbol of logical representation, which specifies that the statement within its range is true for everything or every instance of a particular thing. The Universal quantifier is represented by a symbol ∀, which resembles an inverted A.

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

- **For all x**
- **For each x**

- **For every x.**

Example:

All man drink coffee.

∀x man(x) → drink (x, coffee).

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

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

Existential Quantifier:

Existential quantifiers are the type of quantifiers, which express that the statement within its scope is true for at least one instance of something.It is denoted by the logical operator ∃, which resembles as inverted E. When it is used with a predicate variable then it is called as an existential quantifier.

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.

Note:

- In Existential quantifier we always use AND or Conjunction symbol (∧).
- The main connective for universal quantifier **∀** is implication →.
- The main connective for existential quantifier **∃** is and **∧**.

Properties of Quantifiers:

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

Some Examples of FOL using quantifier:

1. All birds fly.

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

.

2. Every man respects his parent.

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

3. Some boys play cricket.

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

4. Not all students like both Mathematics and Science.

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

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

5. Only one student failed in Mathematics

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

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

2.1.4 Inference Techniques in Logical Agents

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

Techniques:

- Forward Chaining
- Backward Chaining
- Resolution

Forward chaining: Forward chaining is the process where it matches the set of conditions and infer results from these conditions. It is data driven approach using which it reaches (infers) to the goal condition.

Example: Given A (is true)
B->C
A->B
C->D
Prove D is also true.
Solution: Starting from A, A is true then B is true (A->B)
B is true then C is true (B->C)
C is true then D is true Proved (C->D)

Backward chaining: Backward chaining is the process where it performs backward search from goal to the conditions used to get the goal. It is goal driven approach using which it reaches to the initial condition.

Example: Given A (is true)
B->C
A->B
C->D
Prove D is also true.
Solution: Starting from D,
Let D is true then C is true (C->D)
C is true then B is true (B->C)
B is true then A is true Proved (A->B)

2.1.5 Resolution in FOL:

Resolution is a theorem proving technique that proceeds by building refutation proofs, i.e., proofs by contradictions. It was invented by a Mathematician John Alan Robinson in the year 1965.Resolution is used, if there are various statements are given, and we need to prove a conclusion of those statements. Unification is a key concept in proofs by resolutions. Resolution is a single inference rule which can efficiently operate on the conjunctive normal form or clausal form.

- **Clause:** Disjunction of literals (an atomic sentence) is called a clause. It is also known as a unit clause.
- **Conjunctive Normal Form:** A sentence represented as a conjunction of clauses is said to be conjunctive normal form or CNF.

Steps in Resolution:

- Conversion of facts into first-order logic.
- Negate the statement which needs to prove (proof by contradiction)
- Convert FOL statements into CNF
- Draw resolution graph (unification).

To better understand all the above steps, we will take an example in which we will apply resolution.

Example:

Facts:

All peoples who are graduating are happy
All happy people smile
Some one is graduating
Prove that "Is someone smiling?" using resolution

1. Conversion of Facts into first order logic:

- All peoples who are graduating are happy
  ∀x(Graduating(x) ⇒ happy(x))
- All happy people smile
  ∀x(happy(x) ⇒ smile(x))
- Some one is graduating
  ∃x graduating(x)
- "Is someone smiling"
  ∃x smile(x)

2. Negate the statement which needs to prove (proof by contradiction)

- We need to proof "Is someone smiling".
- So negate the statement (proof by contradiction)
  ¬ ∃x smile(x)

3. Convert FOL statements into CNF

Steps to convert FOL to CNF

- Eliminate Implication
- Standardize variable
- Move Negation inwards

- Skolemization
- Drop Universal Quantifier

Eliminate Implication

$\alpha \Rightarrow \beta \equiv \neg \alpha \lor \beta$

$\alpha \Leftrightarrow \beta \equiv (\alpha \Rightarrow \beta) \land (\beta \Rightarrow \alpha)$

Example:

Eliminate Implication: $\alpha \Rightarrow \beta \equiv \neg \alpha \lor \beta$

- $\forall x(\neg \text{Graduating}(x) \lor \text{happy}(x))$
- $\forall x(\neg \text{happy}(x) \lor \text{smile}(x))$
- $\exists x\ \text{graduating}(x)$
- $\neg \exists x\ \text{smile}(x)$

Standardize variable

$\forall x\ \text{regular}(x)$	$\forall x\ \text{regular}(x)$
$\exists x\ \text{busy}(x)$	$\exists y\ \text{busy}(y)$
$\exists x\ \text{attentive}(x)$	$\exists z\ \text{attentive}(z)$

Example:

- $\forall x(\neg \text{Graduating}(x) \lor \text{happy}(x))$
- $\forall y(\neg \text{happy}(y) \lor \text{smile}(y))$
- $\exists z\ \text{graduating}(z)$
- $\neg \ \exists w\ \text{smile}(w)$

Move Negation inwards

- $\neg (\forall x\ P(x)) \equiv \exists x\ \neg P(x)$
- $\neg (\exists\ x\ P(x)) \equiv \forall x\ \neg P(x)$
- $\neg (\alpha \lor \beta) \equiv \neg \alpha \land \neg \beta$
- $\neg (\alpha \land \beta) \equiv \neg \alpha \lor \neg \beta$
- $\neg (\neg \alpha) \equiv \alpha$

Example:

- $\forall x(\neg \text{Graduating}(x) \lor \text{happy}(x))$
- $\forall y(\neg \text{happy}(y) \lor \text{smile}(y))$
- $\exists z\ \text{graduating}(z)$
- $\forall w\ \neg\ \text{smile}(w)$

Skolemization: Removing of Existential Quantifier and replacing it by Skolem Constants

$\exists(y)\ \text{busy}(y)$

$\exists(x)\ \text{attentive}(x)$

After Skolemization it changes into:

busy(A)

attentive(B)

Example:

$\forall x(\neg \text{Graduating}(x) \lor \text{happy}(x))$

∀y(¬happy(y) ∨ smile(y))

After Skolemization it changes into:
graduating(A)
∀w ¬ smile(w)

Drop Universal Quantifier:
∀(y) busy(y)
∀(x) attentive(x)
After dropping it changes into:
busy(y)
attentive(x)

Example:

- ¬ Graduating(x) ∨ happy(x)
- ¬happy(y) ∨ smile(y)
- graduating(A)
- ¬ smile(w)

Now the sentences are in CNF

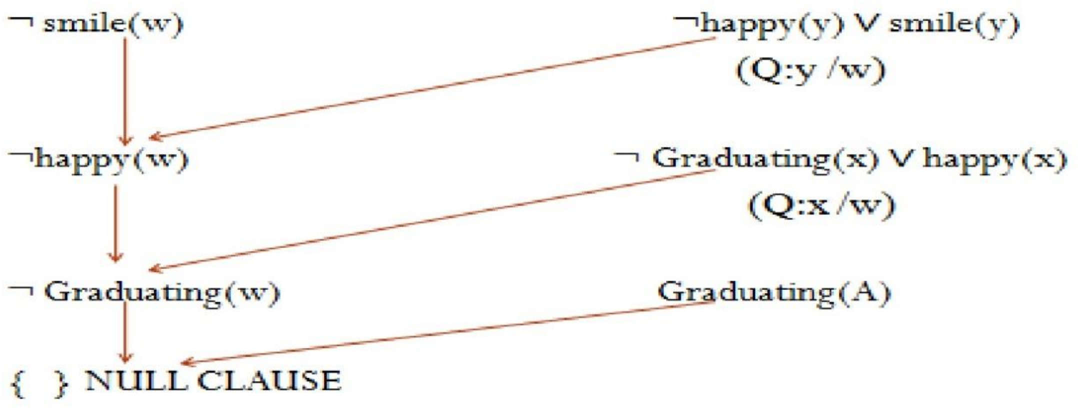
4. Resolution Graph

- If fact F is to be proved then it start with ¬F
- It contradicts all other rules in KB
- The process stop when it returns Null clause

Example:

- ¬ Graduating(x) ∨ happy(x)
- ¬happy(y) ∨ smile(y)
- graduating(A)
- ¬ smile(w)

Resolution Graph:



Hence Someone is Smiling

2.2.1 Semantic Network:

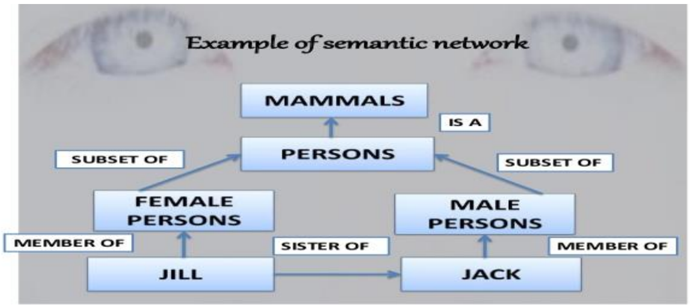
Semantic nets were originally proposed in the early 1960 by M. Ross Quillian to represent the meaning of English words. Semantic networks are alternative of predicate logic for knowledge representation. In Semantic networks, we can represent our knowledge in the form of graphical networks. This network consists of nodes representing objects and arcs which describe the relationship between those objects. Semantic networks can categorize the object in different forms and can also link those objects. Semantic networks are easy to understand and can be easily extended.

Representation:

Semantic network representation consists of mainly two types of relations:

1. IS-A relation (Inheritance)
2. Kind-of-relation

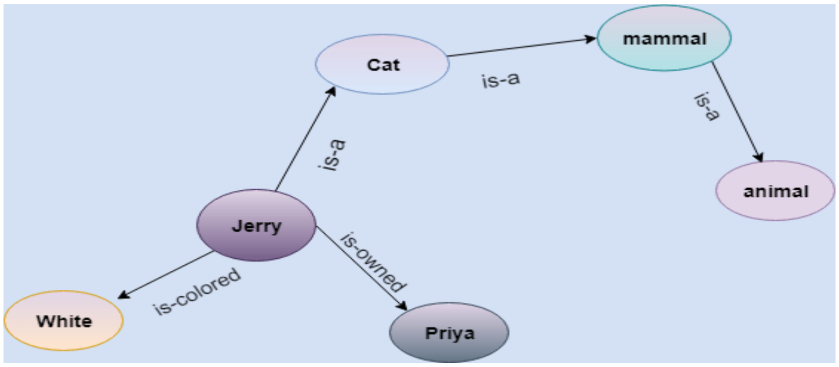
Example 1: Following are some statements which we need to represent in the form of nodes and arcs.



Example 2:

Statements

- Jerry is a cat.
- Jerry is a mammal
- Jerry is owned by Priya.
- Jerry is white colored.
- All Mammals are animal.



Advantages of Semantic Networks:

- Semantic networks are a natural representation of knowledge.
- Semantic networks convey meaning in a transparent manner because it is easy to visualize and understand

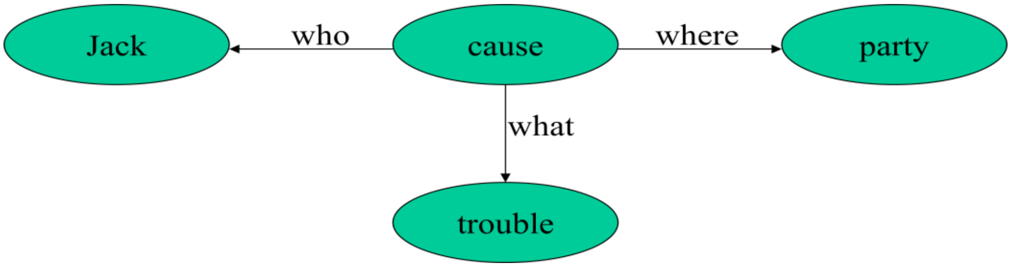
- These networks are simple and easily understandable.
- The knowledge engineer can arbitrarily defined the relationships.
- Related knowledge is easily categorised.
- Efficient in space requirements.
- Node objects represented only once.

Limitations of Semantic Networks:

- The limitations of conventional semantic networks were studied extensively by a number of workers in AI.
- Semantic networks take more computational time at runtime as we need to traverse the complete network tree to answer some questions.
- It might be possible in the worst case scenario that after traversing the entire tree, we find that the solution does not exist in this network.
- Semantic networks try to model human-like memory (Which has 1015 neurons and links) to store the information, but in practice, it is not possible to build such a vast semantic network.
- These types of representations are inadequate as they do not have any equivalent quantifier, e.g., for all, for some, none, etc.
- Many believe that the basic notion is a powerful one and has to be complemented by, for example, **logic** to improve the notion's expressive power and robustness.
- Others believe that the notion of semantic networks can be improved by incorporating **reasoning** used to describe events.
- Binary relations are usually easy to represent, but sometimes is difficult.

  E.g. try to represent the sentence:

  "Jack caused trouble to the party".



- Other problematic statements. . .
  - negation "John does not go fishing";
  - disjunction "John eats pizza or fish and chips";
  - …
- Quantified statements are very hard for semantic nets. E.g.:
  - "Every dog has bitten a delivery boy"
  - "Every dog has bitten every delivery boy"

Solution: Partitioned semantic networks can represent quantified statements.

Partitioned Semantic Networks:

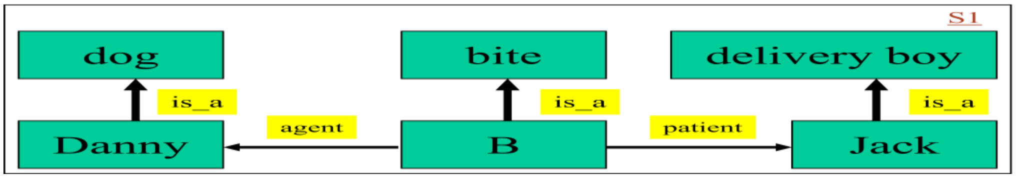
Hendrix developed the so-called partitioned semantic network to represent the difference between the description of an individual object or process and the description of a set of objects. The set description involves quantification. Hendrix partitioned a semantic network whereby a semantic network, loosely speaking, can be divided into one or more networks for the description of an individual.

The central idea of partitioning is to allow groups, nodes and arcs to be bundled together into units called spaces – fundamental entities in partitioned networks, on the same level as nodes and arcs. Every node and every arc of a network belongs to (or lies in/on) one or more spaces.Some spaces are used to encode 'background information' or generic relations; others are used to deal with specifics called 'scratch' space.

Suppose that we wish to make a specific statement about a dog, Danny, who has bitten a delivery boy, Peter:

"Danny the dog bit Jack the delivery boy"

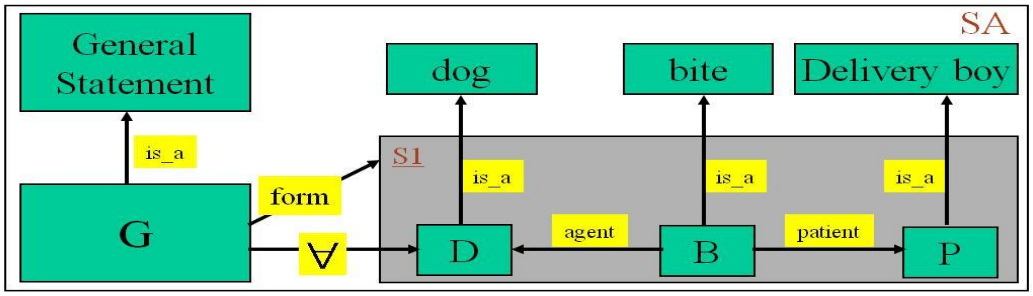
Hendrix’s Partitioned network would express this statement as an ordinary semantic network:



Suppose that we now want to look at the statement:

"Every dog has bitten a delivery boy"

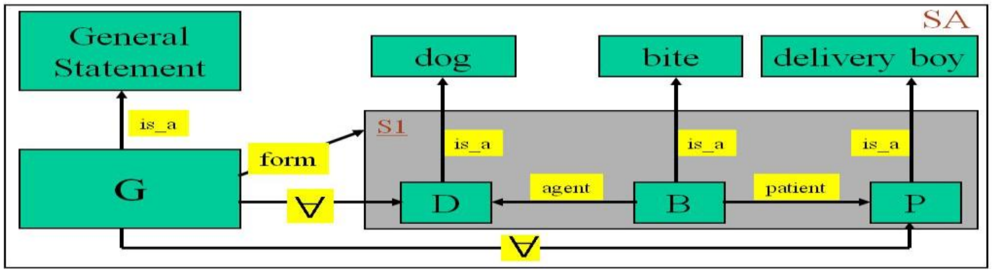
∀



Hendrix partitioned semantic network now comprises two partitions SA and S1. Node G is an **instance** of the special class of general statements about the world comprising link statement, **form**, and one **universal quantifier**

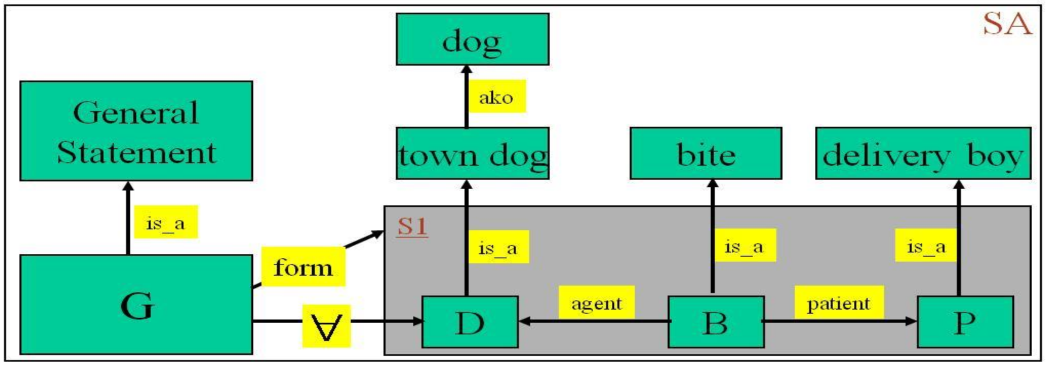
Suppose that we now want to look at the statement:

"Every dog has bitten every delivery boy"



Suppose that we now want to look at the statement:

"Every dog in town has bitten the delivery boy"



Note: 'ako' = 'A Kind Of'

The partitioning of a semantic network renders them more logically adequate, in that one can distinguish between individuals and sets of individuals and indirectly more heuristically adequate by way of controlling the search space by delineating semantic networks.

Hendrix's partitioned semantic networks-oriented formalism has been used in building natural language front-ends for data bases and for programs to deduct information from databases.

2.2.2 Frames:

A frame is a record like structure which consists of a collection of attributes and its values to describe an entity in the world. A frame is analogous to a record structure, corresponding to the fields and values of a record are the slots and slot fillers of a frame

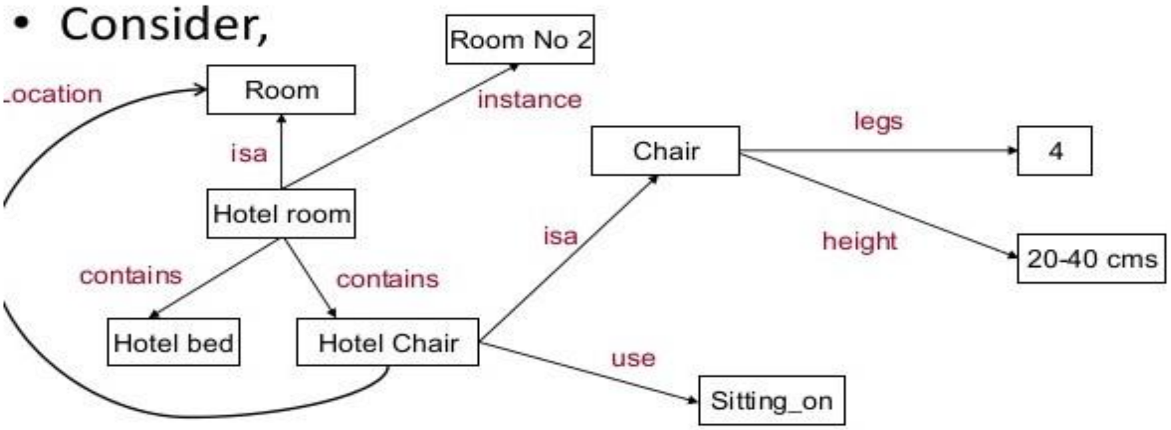
Frames are the AI data structure which divides knowledge into substructures by representing stereotypes situations. It consists of a collection of slots and slot values. These slots may be of any type and sizes. Slots have names and values which are called facets.

Facets: The various aspects of a slot is known as **Facets**. Facets are features of frames which enable us to put constraints on the frames.
Example: IF-NEEDED facts are called when data of any particular slot is needed.

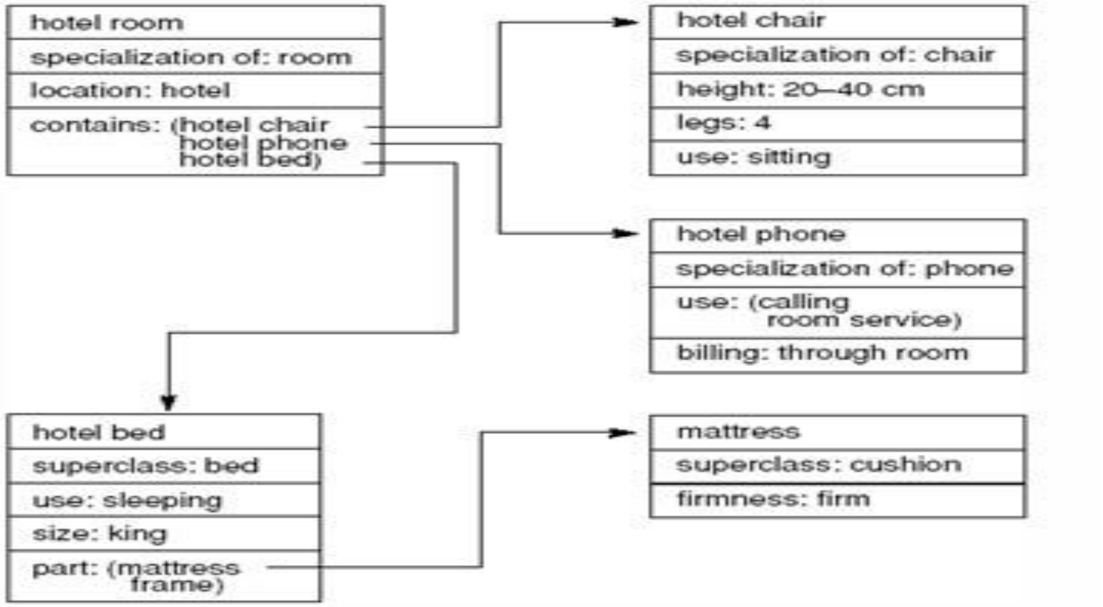
A frame may consist of any number of slots, and a slot may include any number of facets and facets may have any number of values. A frame is also known as **slot-filter knowledge representation** in artificial intelligence.

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

Graphs Representation of Hotel Room:



Frame Description of Hotel Room:



hotel room
specialization of: room
location: hotel
contains: (hotel chair
hotel phone
hotel bed)

hotel chair
specialization of: chair
height: 20–40 cm
legs: 4
use: sitting

hotel phone
specialization of: phone
use: (calling
room service)
billing: through room

hotel bed
superclass: bed
use: sleeping
size: king
part: (mattress
frame)

mattress
superclass: cushion
firmness: firm

Advantages of Frame representation:

- The frame knowledge representation makes the programming easier by grouping the related data.
- The frame representation is comparably flexible and used by many applications in AI.
- It is very easy to add slots for new attribute and relations.

- It is easy to include default data and to search for missing values.
- Frame representation is easy to understand and visualize.

Limitations of Frame representation:

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

2.3.1 Planning Agent:

- An agent interacts with the world via perception and actions
- Perception involves sensing the world and assessing the situation and creating some internal representation of the world
- Actions are what the agent does in the domain. Planning involves reasoning about actions that the agent intends to carry out
- *Planning* is the reasoning side of acting
- This reasoning involves the representation of the world that the agent has, as also the representation of its actions.
- Hard constraints where the objectives *have to* be achieved completely for success
- The objectives could also be soft constraints, or *preferences*, to be achieved as much as possible

Planning:

Planning is arranging a sequence of actions to achieve a goal

○ or

Planning is the task of coming up with a sequence of actions that will achieve the goal

Planning Problem;

- Find a sequence of actions that achieves a given goal when executed from a given initial world state
- That is, given
  - a set of operator descriptions defining the possible primitive actions by the agent,
  - an initial state description, and
  - a goal state description or predicate,
  - compute a plan, which is
  - a sequence of operator instances which after executing them in the initial state changes the world to a goal state
- Goals are usually specified as a conjunction of goals to be achieved

2.3.2: Planning Vs Problem Solving:

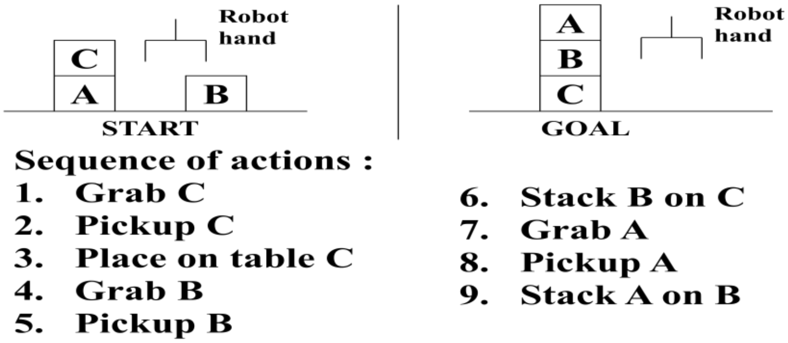
- Planning and problem solving methods can often solve the same sorts of problems
- Planning is more powerful and efficient because of the representations and methods used
- States, goals, and actions are decomposed into sets of sentences (usually in first-order logic)
- Search often proceeds through *plan space* rather than *state space* (though there are also state-space planners)
- Sub goals can be planned independently, reducing the complexity of the planning problem

Example: Block World problem

Block world problem assumptions

- Square blocks of same size
- Blocks can be stacked one upon another.
- Flat surface (table) on which blocks can be placed.
- Robot arm that can manipulate the blocks. It can hold only one block at a time.

In block world problem, the state is described by a set of predicates representing the facts that were true in that state. One must describe for every action, each of the changes it makes to the state description. In addition, some statements that everything else remains unchanged is also necessary.



Sequence of actions :

1. Grab C
2. Pickup C
3. Place on table C
4. Grab B
5. Pickup B
6. Stack B on C
7. Grab A
8. Pickup A
9. Stack A on B

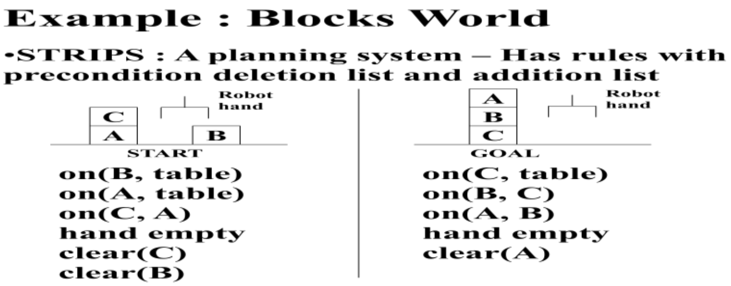
2.3.3 STRIPS Mechanism:

One such mechanism was used in early robot problem solving system named STRIPS (developed by Fikes, 1971).

In this approach, each operation is described by three lists.

- Pre_Cond list contains predicates which have to be true before operation.
- ADD list contains those predicates which will be true after operation
- DELETE list contain those predicates which are no longer true after operation

Predicates not included on either of these lists are assumed to be unaffected by the operation. Frame axioms are specified implicitly in STRIPS which greatly reduces amount of information stored.


Example : Blocks World

•STRIPS : A planning system – Has rules with precondition deletion list and addition list

on(B, table)
on(A, table)
on(C, A)
hand empty
clear(C)
clear(B)

on(C, table)
on(B, C)
on(A, B)
hand empty
clear(A)

Rules:

- ***R1 :*** *pickup(x)*

  Precondition & Deletion List: hand empty, on(x,table), clear(x)

  Add List: holding(x)

- ***R2 :** putdown(x)*
  Precondition & Deletion List: holding(x)
  Add List: hand empty, on(x,table), clear(x)

- ***R3 :** stack(x,y)*
  Precondition & Deletion List: holding(x), clear(y)
  Add List: on(x,y), clear(x), hand empty

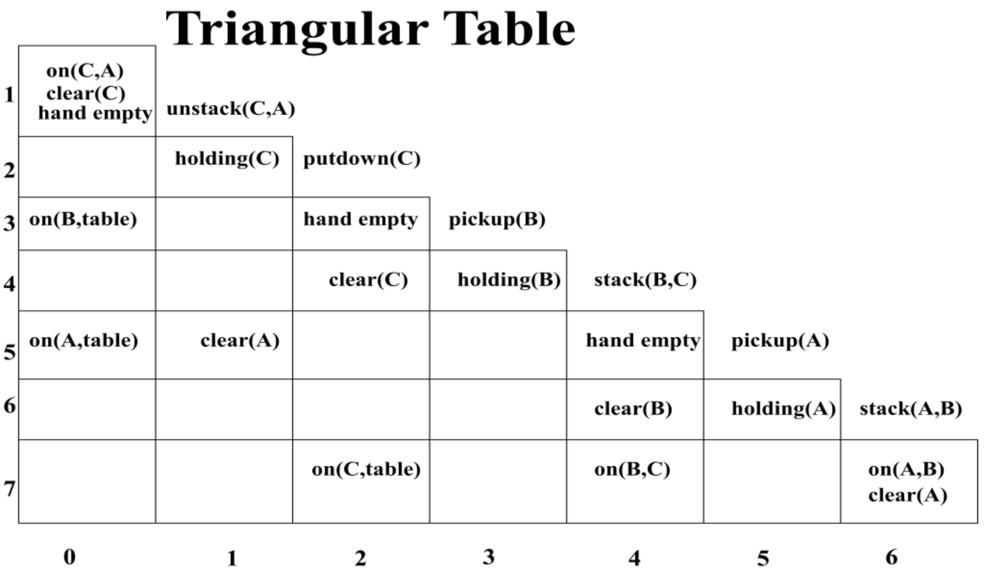
- ***R4 :** unstack(x,y)*
  Precondition & Deletion List: on(x,y), clear(x), hand empty
  Add List: holding(x), clear(y)

Plan for the block world problem:

For the given problem, Start → Goal can be achieved by the following sequence :

1. Unstack(C,A)
2. Putdown(C)
3. Pickup(B)
4. Stack(B,C)
5. Pickup(A)
6. Stack(A,B)

Execution of a plan: achieved through a data structure called Triangular Table.



Triangular Table

1	on(C,A)
	clear(C)
	hand empty
		unstack(C,A)					

2		holding(C)	putdown(C)				
3	on(B,table)		hand empty	pickup(B)			
4			clear(C)	holding(B)	stack(B,C)		
5	on(A,table)	clear(A)			hand empty	pickup(A)	
6					clear(B)	holding(A)	stack(A,B)

7			on(C,table)		on(B,C)		on(A,B)
							clear(A)

	0	1	2	3	4	5	6

- For n operations in the plan, there are :
  - (n+1) rows : 1 → n+1
  - (n+1) columns : 0 → n
- At the end of the i^{th} row, place the i^{th} component of the plan.
- The row entries for the i^{th} step contain the pre-conditions for the i^{th} operation.
- The column entries for the j^{th} column contain the add list for the rule on the top.
- The $<i,j>$ th cell (where $1 \le i \le n+1$ and $0 \le j \le n$) contain the pre-conditions for the i^{th} operation that are added by the j^{th} operation.

- The first column indicates the starting state and the last row indicates the goal state.

Goal Stack Planning:

Goal Stack planning (in short GSP) is one of the most important and simplest planning algorithms, which is specifically used by **STRIPS.** The stack is used in an algorithm to hold the action and satisfy the goal.

A knowledge base is used to hold the current state, actions. This approach uses a Stack for plan generation. The stack can contain Sub-goal and actions described using predicates. The Sub-goals can be solved one by one in any order. Goal stack is similar to a node in a search tree, where the branches are created if there is a choice of an action.

Goal Stack planning algorithm:

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

	Rules	Precondition	Action
1	*pickup(A)*	hand empty, on(A,table), clear(A)	Holding(A)
2	*putdown(A)*	Holding(A)	hand empty, on(A,table), clear(A)
3	*Stack(A,B)*	holding(A), clear(B)	on(A,B), clear(A), hand empty
4	*Unstack(A,B)*	on(A,B), clear(A), hand empty	holding(A), clear(B)

2.3.4 Total Order planning:

- Forward / backward state-space searches are forms of totally ordered plan search.
- Explore only strictly linear sequences of actions, directly connected to the start or goal state.
- Cannot take advantages of problem decomposition

Partial Order planning:

- A **linear planner** builds a plan as a **totally ordered sequence** of plan steps
- A **non-linear planner (aka partial-order planner)** builds up a plan as a set of steps with some temporal constraints
  - constraints like S1<S2 if step S1 must come before S2.
- One **refines** a partially ordered plan (POP) by either: