U18PCCS501 ARTIFICIAL INTELLIGENCE
UNIT I
PROBLEMS AND SEARCH

What is artificial intelligence? - Problems, problem spaces and search – Searching strategies- Uninformed Search- breadth first search, depth first search, uniform cost seart, depth limited search, iterative deepening search, bidirectional search - Informed Search- Best first search, Greedy Best first search , A* search – Constraint satisfaction problem , Local searching strategies.

What is Artificial Intelligence?

The exciting new effort to make computers think … *machines with minds,* in the full literal sense. ***Haugeland, 1985***

The study of how to make computers do things at which, at the moment, people are better. ***Rich & Knight, 1991***

Intelligence:

- "the capacity to learn and solve problems" (Websters dictionary)
- in particular,
  - ✓ *the ability to solve novel problems*
  - ✓ *the ability to act rationally*
  - ✓ *the ability to act like humans*
  - ✓ Artificial Intelligencebuild and understand intelligent entities or agents
- 2 main approaches: "engineering" versus "cognitive modeling"

What is Artificial Intelligence? (John McCarthy, Stanford University)

What is artificial intelligence?

It is the science and engineering of making intelligent machines, especially intelligent computer programs. It is related to the similar task of using computers to understand human intelligence, but AI does not have to confine itself to methods that are biologically observable.

Yes, but what is intelligence?

Intelligence is the computational part of the ability to achieve goals in the world. Varying kinds and degrees of intelligence occur in people, many animals and some machines.

Isn't there a solid definition of intelligence that doesn't depend on relating it to human intelligence?

Not yet. The problem is that we cannot yet characterize in general what kinds of computational procedures we want to call intelligent. We understand some of the mechanisms of intelligence and not others.

What is artificial intelligence?

- **Artificial Intelligence** is the branch of computer science concerned with makingcomputers behave like humans.

- Major AI textbooks define artificial intelligence as "the study and design of intelligent agents," where an **intelligent agent** is a system that **perceives** its **environment** and **takes actions** which maximize its chances of success.

- **John McCarthy**, who coined the term in 1956, defines it as "the science andengineering of making intelligent machines, especially intelligent computer programs."

- The definitions of AI according to some text books are categorized into four approaches and are summarized in the table below :

System that think like Humans	**System that think rationally**
"The exciting new effort to make computers think… machines with minds, in the full and literal sense." (Haugeland,1985)	"The study of mental faculties through the use of computer models" (Charniak and McDermont,1985)
Systems that act like humans	**Systems that act rationally**
"The art of creating machines that performs functions that require intelligence when performed by people." (Kurzweil,1990)	"Computational intelligence is the study of the design of intelligent agents." (Poole et al.,1998)

Applications of Artificial Intelligence:

- **Autonomous planning and scheduling:**

A hundred million miles from Earth, NASA's Remote Agent program becamethe first on-board autonomous planning program to control the scheduling of operations for a spacecraft (Jonsson *et* al., 2000). Remote Agent generated plans from high-level goals specified from the ground, and it monitored the operation of the spacecraft as the plans were executed-detecting, diagnosing,

- **Game playing:**

IBM's Deep Blue became the first computer program to defeat the world champion in a chess match when it bested Garry Kasparov by a score of 3.5 to 2.5 in an exhibition match (Goodman and Keene, 1997).

- **Autonomous control:**

The ALVINN computer vision system was trained to steer a car to keep it following a lane. It was placed in CMU's NAVLAB computer-controlled minivan and used to navigate across the United States-for 2850 miles it was in control of steering the vehicle 98% of the time.

- **Diagnosis:**

Medical diagnosis programs based on probabilistic analysis have been able to perform at the level of an expert physician in several areas of medicine.

- **Logistics Planning:**

During the Persian Gulf crisis of 1991, U.S. forces deployed a Dynamic Analysis and Replanning Tool, DART (Cross and Walker, 1994), to do automated logistics planning and scheduling for transportation. This involved up to 50,000 vehicles, cargo, and people at a time, and had to account for starting points, destinations, routes, and conflict resolution among all parameters. The AI planning techniques allowed a plan to be generated in hours that would have taken weeks with older methods. The Defense Advanced Research Project Agency (DARPA) stated that this single application more than paid back DARPA's 30-year investment in AI.

- **Robotics:**

  Many surgeons now use robot assistants in microsurgery. HipNav (DiGioia *et* al., 1996) is a system that uses computer vision techniques to create a three-dimensional model of a patient's internal anatomy and then uses robotic control to guide the insertion of a hip replacement prosthesis.

- **Language understanding and problem solving:**

  PROVERB (Littman *et al.,* 1999) is a computer program that solves crossword puzzles better than most humans, using constraints on possible word fillers, a large database of past puzzles, and a variety of information sources including dictionaries and online databases such as a list of movies and the actors that appear in them.

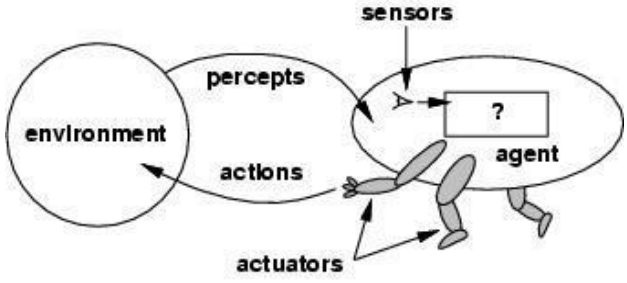
AGENTS:

Rationality concept can be used to develop a smallest of design principle for building successful agents; these systems are reasonably called as Intelligent.

Agents and environments:

An **agent** is anything that can be viewed as perceiving its **environment** through **sensors** and SENSOR acting upon that environment through **actuators.** This simple idea is illustrated in Figure.

- A human agent has eyes, ears, and other organs for sensors and hands, legs, mouth, and other body parts for actuators.
- A robotic agent might have cameras and infrared range finders for sensors and various motors for actuators.
- A software agent receives keystrokes, file contents, and network packets as sensory inputs and acts on the environment by displaying on the screen, writing files, and sending network packets.



Percept

We use the term **percept** to refer to the agent's perceptual inputs at any given instant.

Percept Sequence

An agent's **percept sequence** is the complete history of everything the agent has ever perceived.

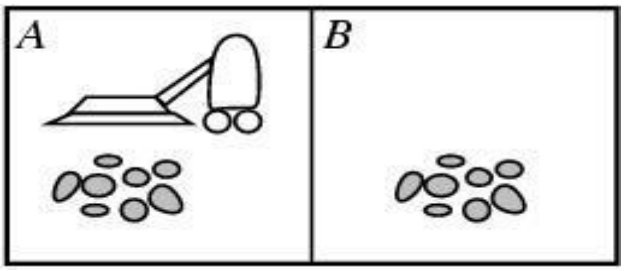
Agent function

Mathematically speaking, we say that an agent's behavior is described by the **agent function** that maps any given percept sequence to an action.

Agent program

$$f : \mathcal{P}^* \rightarrow \mathcal{A}$$

- The agent function for an artificial agent will be implemented by an **agent program.**
- It is important to keep these two ideas distinct.
- The agent function is an abstract mathematical description;
- the agent program is a concrete implementation, running on the agent architecture.
- To illustrate these ideas, we will use a very simple example-the vacuum-cleaner world shown in Figure.
- This particular world has just two locations: squares A and B.
- The vacuum agent perceives which square it is in and whether there is dirt in the square.
- It can choose to move left, move right, suck up the dirt, or do nothing.
- One very simple agent function is the following:
- if the current square is dirty, then suck, otherwise,
- it move to the other square.
- A partial tabulation of this agent function is shown in Figure.



Agent function

Percept Sequence	Action
[A, Clean]	Right
[A, Dirty]	Suck
[B, Clean]	Left
[B, Dirty]	Suck
[A, Clean], [A, Clean]	Right
[A, Clean], [A, Dirty]	Suck
…..	…..

Agent program

function Reflex-VACUUM-AGENT ([locations, status]) **returns an action**

if status = Dirty **then return** Suck
else if location = A **then return** Right
elseif location = B **then return** Left

Good Behavior: The concept of Rationality

- A **rational agent** is one that does the right thing-conceptually speaking; every entry in the table for the agent function is filled out correctly.
- Obviously, doing the right thing is better than doing the wrong thing.
- The right action is the one that will cause the agent to be most successful.

Performance measures

- **A performance measure** embodies the **criterion for success** of an agent's behavior.
- When an agent is plunked down in an environment, it generates a sequence of actions according to the percepts it receives.
- This sequence of actions causes the environment to go through a sequence of states.
- If the sequence is desirable, then the agent has performed well.

Rationality

What is rational at any given time depends on four things:

- The performance measure that defines the criterion of success.
- The agent's prior knowledge of the environment.
- The actions that the agent can perform. o The agent's percept sequence to date.
- This leads to a **definition of a rational agent:**

For each possible percept sequence, a rational agent should select an action that is expected to maximize its performance measure, given the evidence provided by the percept sequence and whatever built-in knowledge the agent has.

Omniscience, learning, and autonomy

- An **omniscient agent** knows the actual outcome of its actions and can act accordingly; but omniscience is impossible in reality.
- Doing actions in order to modify future percepts-sometimes called **informationgathering**-is an important part of rationality.
- Our definition requires a rational agent not only to gather information, but also to **learn** as much as possible from what it perceives.
- To the extent that an agent relies on the prior knowledge of its designer rather than on its own percepts, we say that the agent lacks autonomy.
- A rational agent should be **autonomous**-it should learn what it can to compensate for partial or incorrect prior knowledge.

Task environments

- We must think about **task environments,** which are essentially the "**problems**" to which rational agents are the "**solutions**."

Specifying the task environment

- The rationality of the simple vacuum-cleaner agent, needs specification of
  - the performance measure
  - the environment
  - the agent's actuators
  - Sensors.

PEAS

- All these are grouped together under the heading of the **task environment.**
- We call this the **PEAS** (Performance, Environment, Actuators, Sensors) description.
- In designing an agent, the first step must always be to specify the task environment as fully as possible.
- The following table shows PEAS description of the task environment for an automated taxi.

Agent Type	**Performance Measure**	**Environments**	**Actuators**	**Sensors**
Taxi driver	Safe: legal, comfortable trip, maximize Profits	Roads,other traffic,pedestrians, customers	Steering,accelerator, brake, Signal,horn,display	Cameras,sonar, Speedometer,GPS, Odometer,engine sensors,keyboards, accelerometer

The following table shows PEAS description of the task environment for some other agent type.

Agent Type	Performance Measure	Environment	Actuators	Sensors
Medical diagnosis system	Healthy patient, minimize costs, lawsuits	Patient, hospital, staff	Display questions, tests, diagnoses, treatments, referrals	Keyboard entry of symptoms, findings, patient's answers
Satellite image analysis system	Correct image categorization	Downlink from orbiting satellite	Display categorization of scene	Color pixel arrays
Part-picking robot	Percentage of parts in correct bins	Conveyor belt with parts; bins	Jointed arm and hand	Camera, joint angle sensors
Refinery controller	Maximize purity, yield, safety	Refinery, operators	Valves, pumps, heaters, displays	Temperature, pressure, chemical sensors
Interactive English tutor	Maximize student's score on test	Set of students, testing agency	Display exercises, suggestions, corrections	Keyboard entry

Properties of task environments

- **Fully observable vs. partially observable**
- **Deterministic vs. stochastic**
- **Episodic vs. sequential**
- **Static vs. dynamic**
- **Discrete vs. continuous**
- **Single agent vs. multiagent**

Fully observable vs. **partially observable.**

- If an agent's sensors give it access to the complete state of the environment at each point in time, then we say that the task environment is fully observable.
- A task environment is effectively fully observable if the sensors detect all aspects that are *relevant* to the choice of action;
- 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.

Deterministic vs. **stochastic.**

- If the next state of the environment is completely determined by the current state and the action executed by the agent, then we say the environment is deterministic;
- Otherwise, it is stochastic.

Episodic vs. **sequential**

- In an **episodic task environment**, the agent's experience is divided into atomic episodes.
- Each episode consists of the agent perceiving and then performing a single action. Crucially, the next episode does not depend on the actions taken in previous episodes.
- For example, an agent that has to spot defective parts on an assembly line bases each decision on the current part, regardless of previous decisions;
- In **sequential environments**, on the other hand, the current decision Could affect all future decisions.
- Chess and taxi driving are sequential:

Discrete vs. **continuous.**

- The discrete/continuous distinction can be applied to the *state* of the environment, to the way *time* is handled, and to the *percepts* and *actions* of the agent.
- For example, a discrete-state environment such as a chess game has a finite number of distinct states.
- Chess also has a discrete set of percepts and actions.
- Taxi driving is a continuous- state and continuous-time problem:
- The speed and location of the taxi and of the other vehicles sweep through a range of continuous values and do so smoothly over time.
- Taxi-driving actions are also continuous (steering angles, etc.)

Single agent vs. **multiagent.**

- An agent solving a crossword puzzle by itself is clearly in a single-agent environment,
- Where as an agent playing chess is in a two-agent environment.
- Multiagent is further classified in to two ways

  - Competitive multiagent environment
  - Cooperative multiagent environment

Agent programs

- The job of Artificial Intelligence is to design the agent program that implements the agent function mapping percepts to actions
- The agent program will run in an architecture
- An architecture is a computing device with physical sensors and actuators
- Where Agent is combination of Program and Architecture

Agent = Program + Architecture

- An agent program takes the current percept as input while the agent function takes the entire percept history
- Current percept is taken as input to the agent program because nothing more is available from the environment
- The following TABLE-DRIVEN_AGENT program is invoked for each new percept and returns an action each time

Function TABLE-DRIVEN_AGENT (percept) **returns** an action

static: percepts, a sequence initially empty
table, a table of actions, indexed by percept sequence

append percept to the end of percepts
action LOOKUP(percepts, table)
return action

Drawbacks:

- **Table lookup** of percept-action pairs defining all possible condition-action rules necessary to interact in an environment
- **Problems**
  - Too big to generate and to store (Chess has about 10^120 states, for example)
  - No knowledge of non-perceptual parts of the current state
  - Not adaptive to changes in the environment; requires entire table to be updated if changes occur
  - Looping: Can't make actions conditional

- Take a long time to build the table
- No autonomy
- Even with learning, need a long time to learn the table entries

Some Agent Types

- **Table-driven agents**
  - use a percept sequence/action table in memory to find the next action. They are implemented by a (large) **lookup table**.

- **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**
  - have **internal state**, which is used to keep track of past states of the world.

- **Agents with goals**
  - 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.

Kinds of Agent Programs

- The following are the agent programs,
  - Simple reflex agents
  - Mode-based reflex agents
  - Goal-based reflex agents
  - Utility-based agents

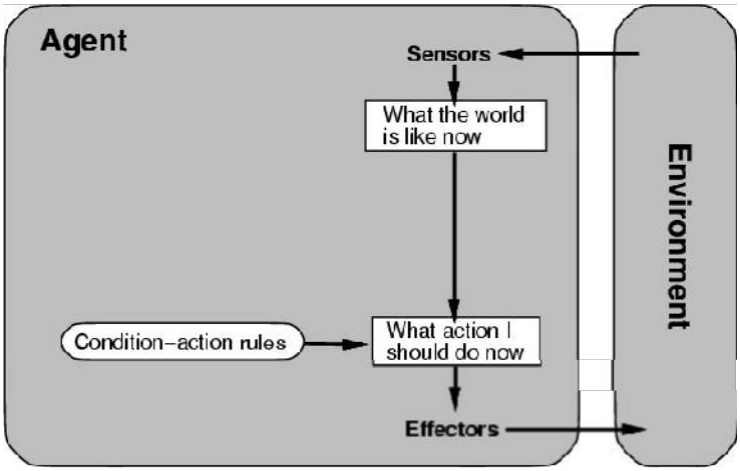
Simple Reflex Agent

- The simplest kind of agent is the **simple reflex agent.**
- These agents select actions on the basis of the *current* percept, ignoring the rest of the percept history.
- For example, the vacuum agent whose agent function is tabulated is given below,
- a simple reflex agent, because its decision is based only on the current location and on whether that contains dirt.
- Select action on the basis of *only the current* percept.E.g. the vacuum-agent
- Large reduction in possible percept/action situations(next page).
- Implemented through *condition-action rules*
- If dirty then suck

A Simple Reflex Agent: Schema

- Schematic diagram of a simple reflex agent.
- The following simple reflex agents, acts according to a rule whose condition matches the current state, as defined by the percept

function SIMPLE-REFLEX-AGENT(*percept*) **returns** an action
static:*rules*, a set of condition-action rules
state □INTERPRET - INPUT(*percept*)
rule □RULE-MATCH(*state*, *rule*)
action □RULE-ACTION[*rule*] return *action*



- The agent program for a simple reflex agent in the two-state vacuum environment.
  function REFLEX-VACUUM-AGENT ([location, status]) return an action
  if status == Dirty then return Suck
  else if location == A then return Right
  else if location == B then return Left

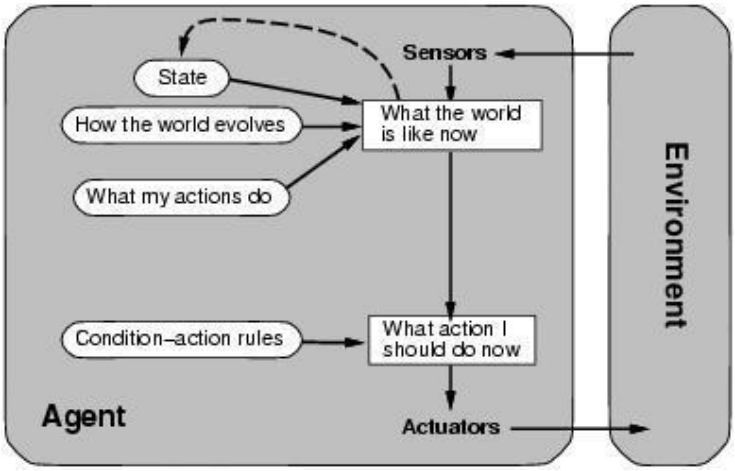
Characteristics

- Only works if the environment is fully observable.
- Lacking history, easily get stuck in infinite loops
- One solution is to randomize actions

Model-based reflex agents

- The most effective way to handle partial observability is for the agent to *keep track ofthe part of the world it can't see now.*
- That is, the agent should maintain some sort of **internal state** that depends on the percept history and thereby reflects at least some of the unobserved aspects of the current state.
- Updating this internal state information as time goes by requires two kinds of knowledge to be encoded in the agent program.
- First, we need some information about how the world evolves independently of the agent
- For example, that an overtaking car generally will be closer behind than it was a moment ago.

- Second, we need some information about how the agent's own actions affect the world
- For example, that when the agent turns the steering wheel clockwise, the car turns to the right or that after driving for five minutes northbound on the freeway one is usually about five miles north of where one was five minutes ago.
- This knowledge about "how the world working - whether implemented in simple Boolean circuits or in complete scientific theories-is called a **model** of the world.
- An agent that uses such a MODEL-BASED model is called a **model-based agent.**
- Schematic diagram of A model based reflex agent

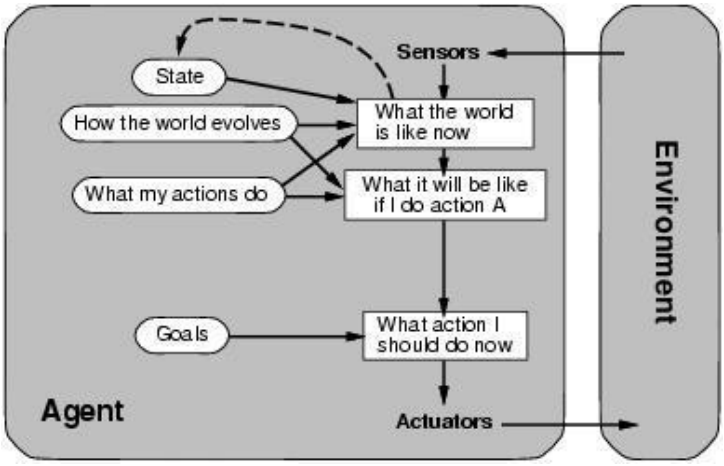


- Model based reflex agent. It keeps track of the current state of the world using an internal model. It then chooses an action in the same way as the reflex agent.

function REFLEX-AGENT-WITH-STATE(*percept*) **returns** an action **static**:*rules*, a set of condition-action rules
state, a description of the current world state
action, the most recent action.
state VPDATE-STATE(*state*, *action*, *percept*)
rule RVLE- MATCH(*state, rule*)
action RVLE-
ACTION[*rule*] return *action*

Goal-based agents

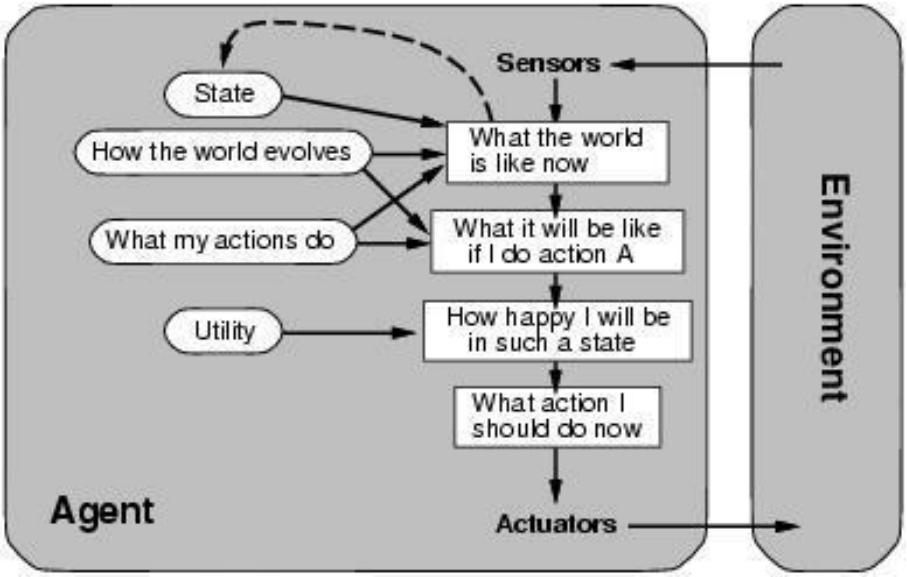
- Knowing about the current state of the environment is not always enough to decide what to do.
- For example, at a road junction, the taxi can turn left, turn right, or go straight on. The correct decision depends on where the taxi is trying to get to.
- In other words, as well as a current state description, the agent needs some sort of **goal** information that describes situations that are desirable.
- For example, being at the passenger's destination.
- The agent program can combine this with information about the results of possible actions (the same information as was used to update internal state in the reflex agent) in order to choose actions that achieve the goal.
- Schematic diagram of the goal-based agent's structure.



Utility-based agents

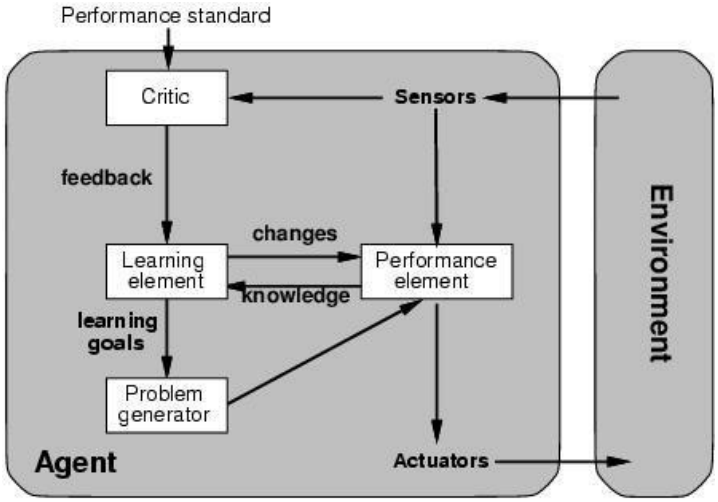
- Goals alone are not really enough to generate high-quality behavior in most environments.
- For example, there are many action sequences that will get the taxi to its destination (thereby achieving the goal) but some are quicker, safer, more reliable, or cheaper than others.
- Goals just provide a crude binary distinction between "happy" and "unhappy" states, whereas a more general **performance measure** should allow a comparison of different world states according to exactly how happy they would make the agent if they could be achieved. Because "happy" does not sound very scientific, the customary terminology is to say that if one world state is preferred to another, then it has higher **utility** for the agent.
- Schematic diagram of a utility-based agents
- It uses a model of the world, along with a utility function that measures its preferences among states of the world.

- Then it chooses the action that leads to the best expected utilit y, where expected utility is computed by averaging over all possible outcome states, weighted by the probability of the outcome.
- Certain goals can be reached in different ways
    - Some are better, have a higher utility
- Utility function maps a (Sequence of) state(S) onto a real number.
- Improves on goal:
    - Selecting between conflicting goals
    - Select appropriately between several goals based on likelihood of Success



Learning Agent

Schematic diagram of Learning Agent



- All agents can improve their performance through learning.
- A learning agent can be divided into four conceptual components, as,
    - Learning element
    - Performance element
    - Critic
    - Problem generator

- The most important distinction is between the **learning element**, which is responsible for making improvements,
- The **performance element**, which is responsible for selecting external actions.
- The performance element is what we have previously considered to be the entire agent: it takes in precepts and decides on actions.
- The learning element uses feedback from the **critic** on how the agent is doing and determines how the performance element should be modified to do better in the future.
- The last component of the learning agent is the **problem generator**.
- It is responsible for suggesting actions that will lead to new and **informative experiences**. But if the agent is willing to explore a little, it might discover much better actions for the long run.
- The problem generator's job is to suggest these **exploratory actions**. This is what scientists do when they carry out experiments.

Summary: Intelligent Agents

- An **agent** perceives and acts in an environment, has an architecture, and is implemented by an agent program.
- Task environment - **PEAS (P**erformance**, E**nvironment, **A**ctuators, **S**ensors**)**
- The most challenging environments are inaccessible, nondeterministic, dynamic, and continuous.
- An **ideal agent** always chooses the action which maximizes its expected performance, given its percept sequence so far.
- An **agent program** maps from percept to action and updates internal state.
    - **Reflex agents** respond immediately to precepts.
        - simple reflex agents
        - model-based reflex agents
    - **Goal-based agents** act in order to achieve their goal(s).
    - **Utility-based agents** maximize their own utility function.
- All agents can improve their performance through **learning.**

Problem Formulation

- An important aspect of intelligence is *goal-based* problem solving.
- The solution of many problems can be described by finding a sequence of actions that lead to a desirable goal.
- Each action changes the *state* and the aim is to find the sequence of actions and states that lead from the initial (start) state to a final (goal) state.
- **A well-defined problem can be described by:**
    - **Initial state**
    - **Operator or successor function** - for any state x returns s(x), the set ofstates reachable from x with one action
    - **State space** - all states reachable from initial by any sequence of actions **Path -** sequence through state space
    - **Path cost** - function that assigns a cost to a path. Cost of a path is the sumof costs of individual actions along the path
    - **Goal test -** test to determine if at goal state
- What is **Search**?
- Search is the systematic examination of states to find path from the start/root state to the goal state.
- The set of possible states, together with *operators* defining their connectivity constitute the *search space*.
- The output of a search algorithm is a solution, that is, a path from the initial state to a state that satisfies the goal test.

Problem-solving agents

- A Problem solving agent is a goal-based agent.
- It decides what to do by finding sequence of actions that lead to desirable states.
- The agent can adopt a goal and aim at satisfying it.
- To illustrate the agent's behavior
- For example where our agent is in the city of Arad, which is in Romania. The agent has to adopt a goal of getting to Bucharest.
- Goal formulation, based on the current situation and the agent's performance measure, is the first step in problem solving.

- The agent's task is to find out which sequence of actions will get to a goal state.
- Problem formulation is the process of deciding what actions and states to consider given a goal.

Example: **Route finding problem**
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:
sequence of cities, e.g., Arad, Sibiu, Fagaras, Bucharest

- Goal formulation and problem formulation

A **problem** is defined by four items:
initial state e.g., "at Arad"
successor function S(x) = set of action-state pairs
e.g., S(Arad) = {[Arad -> Zerind;Zerind],….}
goal test, can be
explicit, e.g., x = at Bucharest"
implicit, e.g., NoDirt(x)
path cost (additive)
e.g., sum of distances, number of actions executed, etc.
c(x; a; y) is the step cost, assumed to be >= 0

A **solution** is a sequence of actions leading from the initial state to a goal state.

Search

- An agent with several immediate options of unknown value can decide what to do b y examining different possible sequences of actions that leads to the states of known value,and then choosing the best sequence.

- The process of looking for sequences actions from the current state to reach the goal state is called **search.**

- The **search algorithm** takes a **problem** as **input** and returns a **solution** in the form of **action sequence.**
- Once a solution is found,the **execution phase** consists of carrying out the recommended action.

- The following shows a simple "formulate,search,execute" design for the agent.

- Once solution has been executed, the agent will formulate a new goal.

- It first formulates a **goal** and a **problem**,searches for a sequence of actions that would solve a problem,and executes the actions one at a time.

function SIMPLE-PROBLEM-SOLVING-AGENT(*percept*)
returns an action
inputs :*percept*, a percept
static:*seq*, an action sequence, initially empty
state, some description of the current world
state *goal*, a goal, initially null
problem, a problem formulation
state VPDATE-STATE(*state, percept*)
if seq is empty **then do**
goal ← FORMVLATE-GOAL(*state*)
problem ← FORMVLATE-PROBLEM(*state, goal*)
seq ← SEARCH(*problem*)
action ← FIRST(*seq*);
seq ← REST(seq)
return *action*

- The agent design assumes the Environment is

Static: The entire process carried out without paying attention to changes thatmight be occurring in the environment.

Observable : The initial state is known and the agent's sensor detects allaspects that are relevant to the choice of action

Discrete : With respect to the state of the environment and percepts andactions so that alternate courses of action can be taken

Deterministic: The next state of the environment is completely determined b ythe current state and the actions executed by the agent. Solutions to the problem are single sequence of actions

An agent carries out its plan with eye closed. This is called an open loop system because ignoring the precepts breaks the loop between the agent and the environment.

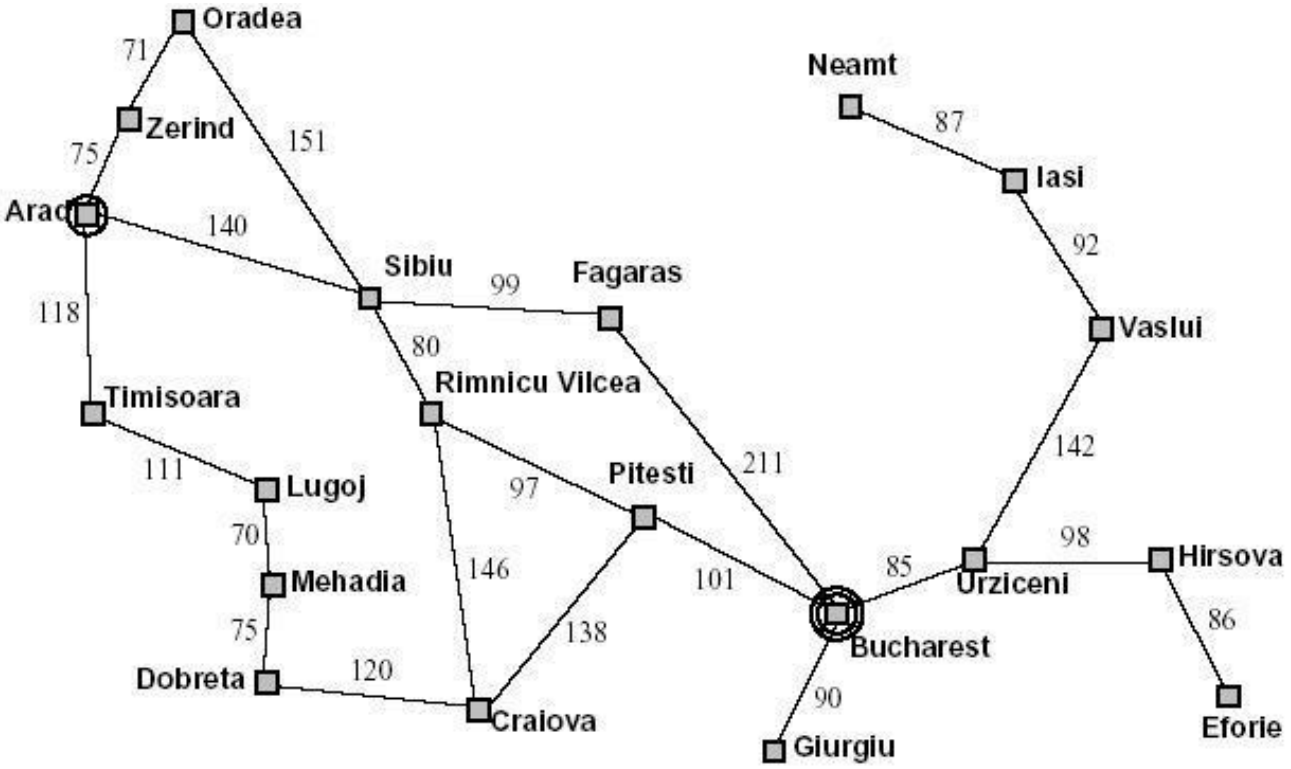
Well-defined problems and solutions

A **problem** can be formally defined by **four components**:

- The **initial state** that the agent starts in . The initial state for our agent of example problem is described by *In(Arad)*
- A **Successor Function** returns the possible **actions** available to the agent.
- Given a state x,SVCCESSOR-FN(x) returns a set of {action,successor} ordered pairs where each action is one of the legal actions in state x,and each successor is a state that can be reached from x by applying the action.
- For example, from the state In(Arad),the successor function for the Romania problem would return

{
[Go(Sibiu),In(Sibiu)],[Go(Timisoara),In(Timisoara)],[Go(Zerind),In(Zerind)] }

- **State Space**: The set of all states reachable from the initial state. The statespace forms a graph in which the nodes are states and the arcs between nodes are actions.
- A **path** in the state space is a sequence of states connected by a sequence of actions.
- The **goal test** determines whether the given state is a goal state. A **path cost** function assigns numeric cost to each action.
- For the Romania problem the cost of path might be its length in kilo meters.
- The **step cost** of taking action a to go from state x to state y is denoted b y c(x,a,y). It is assumed that the step costs are non negative.
- A **solution** to the problem is a path from the initial state to a goal state.
- An **optimal solution** has the lowest path cost among all solutions.



A simplified Road Map of part of Romania

Advantages:

They are easy enough because they can be carried out without further search or planning

The choice of a good abstraction thus involves removing as much details as possible while retaining validity and ensuring that the abstract actions are eas y to carry out.

EXAMPLE PROBLEMS

The problem solving approach has been applied to a vast array of task environments. Some best known problems are summarized below.

They are distinguished as toy or real-world problems

- A T**oy problem** is intended to illustrate various problem solving methods. It can be easily used by different researchers to compare the performance of algorithms.

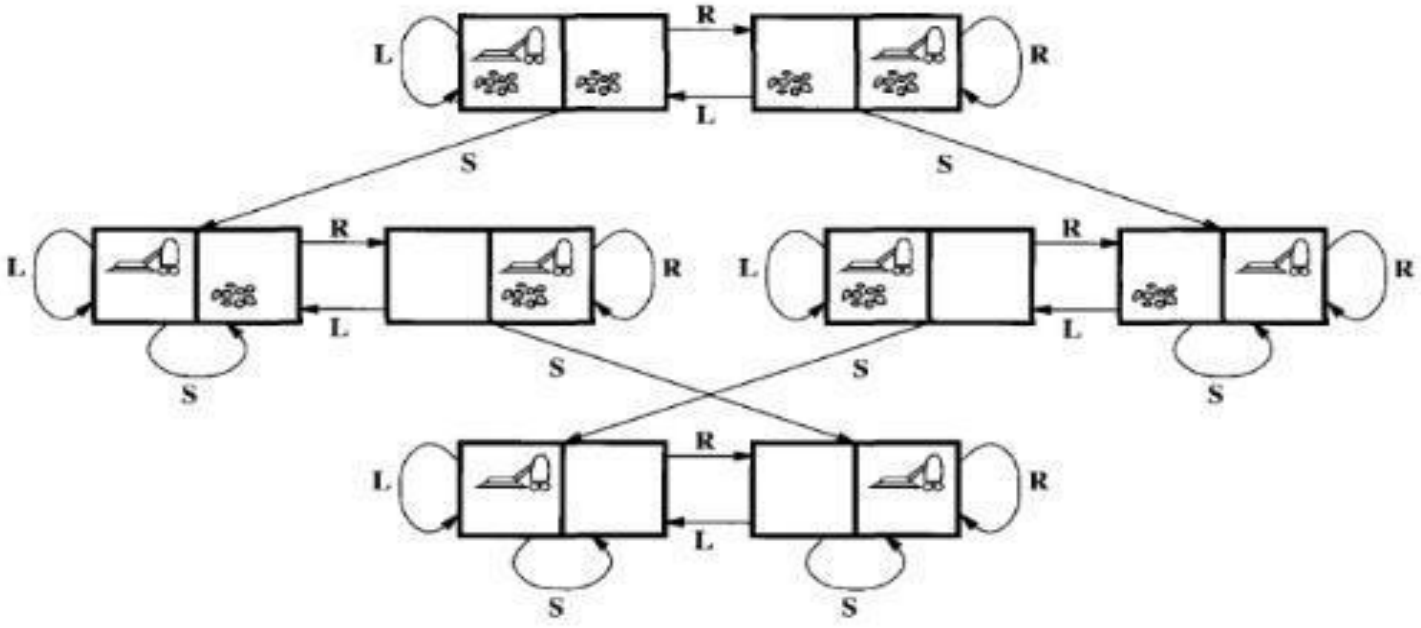
- A R**eal world problem** is one whose solutions people actually care about.

TOY PROBLEMS

Vacuum World Example

- **States**: The agent is in one of two locations., each of which might or might not contain dirt. Thus there are 2 x 2^2 = 8 possible world states.
- **Initial state**: Any state can be designated as initial state.
- **Successor function** : This generates the legal states that results from trying the three actions(left, right, suck). The complete state space is shown in figure 2.3
- **Goal Test** : This tests whether all the squares are clean.
- **Path test** : Each step costs one ,so that the the path cost is the number of steps in the path.

Vacuum World State Space



The state space for the vacuum world.
Arcs denote actions: L = Left,R = Right,S = Suck

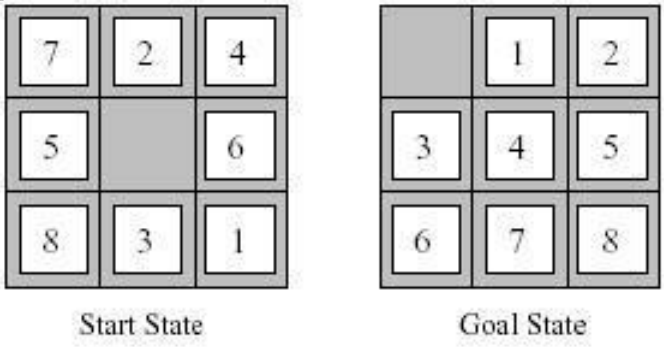
8-puzzle:

- An 8-puzzle consists of a 3x3 board with eight numbered tiles and a blank space.

- A tile adjacent to the blank space can slide into the space. The object is to reach the specific goal state ,as shown in figure

Example: The 8-puzzle

.



A typical instance of 8-puzzle.

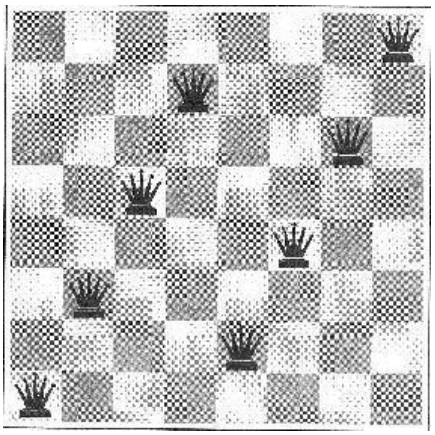
The problem formulation is as follows :

- **States** : A state description specifies the location of each of the eight tiles and theblank in one of the nine squares.
- **Initial state** : Any state can be designated as the initial state. It can be noted that anygiven goal can be reached from exactly half of the possible initial states.
- **Successor function** : This generates the legal states that result from trying the four actions (blank moves Left,Right,Vp or down).
- **Goal Test** : This checks whether the state matches the goal configuration shown in figure 2.4.(Other goal configurations are possible)
- **Path cost**: Each step costs 1,so the path cost is the number of steps in the path.
- **The 8-puzzle** belongs to the family **of sliding-block puzzles**, which are often used astest problems for new search algorithms in AI.
- This general class is known as NP-complete.
- The **8-puzzle** has 9!/2 = 181,440 reachable states and is easily solved.
- The **15 puzzle** (4 x 4 board) has around 1.3 trillion states, an the random instances can be solved optimally in few milli seconds by the best search algorithms.
- The **24-puzzle** (on a 5 x 5 board) has around 10^{25} states ,and random instances are still quite difficult to solve optimally with current machines and algorithms.

8-queens problem

- The goal of 8-queens problem is to place 8 queens on the chessboard such that no queen attacks any other. (A queen attacks any piece in the same row, column or diagonal).
- The following figure shows an attempted solution that fails: the queen in the right most column is attacked by the queen at the top left.
- An **Incremental formulation** involves operators that augments the state description,starting with an empty state.for 8-queens problem,this means each action adds a queen to the state.
- A **complete-state formulation** starts with all 8 queens on the board and move themaround.

o In either case the path cost is of no interest because only the final state counts.



The first incremental formulation one might try is the following :

- **States** : Any arrangement of 0 to 8 queens on board is a state.
- **Initial state** : No queen on the board.
- **Successor function** : Add a queen to any empty square.
- **Goal Test** : 8 queens are on the board,none attacked.

In this formulation, we have 64.63…57 = 3 x 10^{14} possible sequences to investigate.

A better formulation would prohibit placing a queen in any square that is alread y attacked. :

- **States** : Arrangements of n queens (0 <= n < = 8) ,one per column in theleft most columns ,with no queen attacking another are states.

- **Successor function** : Add a queen to any square in the left most emptycolumn such that it is not attacked by any other queen.

This formulation reduces the 8-queen state space from 3 x 10^{14} to just 2057,and solutions are easy to find.

For the 100 queens the initial formulation has roughly 10^{400} states whereas the improved formulation has about 10^{52} states.

This is a huge reduction, but the improved state space is still too big for the algorithms to handle.

REAL WORLD PROBLEMS

- A real world problem is one whose solutions people actually care about.
- They tend not to have a single agreed upon description, but attempt is made to give general flavor of their formulation,
- The following are the some real world problems,

    - Route Finding Problem
    - Touring Problems
    - Travelling Salesman Problem
    - Robot Navigation

ROUTE-FINDING PROBLEM

- Route-finding problem is defined in terms of specified locations and transitions along links between them.

- Route-finding algorithms are used in a variety of applications, such as routing in computer networks, military operations planning, and air line travel planning systems.

AIRLINE TRAVEL PROBLEM

The **airline travel problem** is specifies as follows **:**

- **States :** Each is represented by a location(e.g.,an airport) and the current time.
- **Initial state :**This is specified by the problem.
- **Successor function :**This returns the states resulting from taking any scheduledflight(further specified by seat class and location),leaving later than the current time plus the within-airport transit time,from the current airport to another.
- **Goal Test :** Are we at the destination by some prespecified time?
- **Path cost :** This depends upon the monetary cost,waiting time,flight time,customsand immigration procedures,seat quality,time of dat,type of air plane,frequent-flyer mileage awards, and so on.

TOURING PROBLEMS

- **Touring problems** are closely related to route-finding problems,but with animportant difference.
- Consider for example, the problem, "Visit every city at least once" as shown in Romania map.
- As with route-finding the actions correspond to trips between adjacent cities. The state space, however,is quite different.

  - **Initial state** would be "In Bucharest; visited{Bucharest}".
  - **Intermediate state** would be "In Vaslui; visited{Bucharest,Vrziceni,Vaslui}".
  - **Goal test** would check whether the agent is in Bucharest and all 20cities have been visited.

THE TRAVELLING SALESPERSON PROBLEM (TSP)

- ➔ TSP is a touring problem in which each city must be visited exactly once
- ➔ The aim is to find the shortest tour. The problem is known to be **NP-hard.**
- ➔ Enormous efforts have been expended to improve thecapabilities of TSP algorithms.
- ➔ These algorithms are also used in tasks such as planningmovements of **automatic circuit-board drills** and of **stocking machines** on shop floors.

VLSI layout

A **VLSI layout** problem requires positioning millions of components and connections on a chip to minimize area ,minimize circuit delays,minimize stray capacitances,and maximize manufacturing yield. The layout problem is split into two parts : **cell layout** and **channel routing**.

ROBOT navigation

ROBOT navigation is a generalization of the route-finding problem. Rather than adiscrete set of routes,a robot can move in a continuous space with an infinite set of possible actions and states. For a circular Robot moving on a flat surface,the space is essentially two-dimensional.

When the robot has arms and legs or wheels that also must be controlled,the search space becomes multi-dimensional. Advanced techniques are required to make the search space finite.

AUTOMATIC ASSEMBLY SEQUENCING

The example includes assembly of intricate objects such as electric motors. The aim in assembly problems is to find the order in which to assemble the parts of some objects. If the wrong order is choosen, there will be no way to add some part later without undoing somework already done.

Another important assembly problem is protein design, in which the goal is to find a sequence of

Amino acids that will be fold into a three-dimensional protein with the right properties to cure some disease.

INTERNET SEARCHING

In recent years there has been increased demand for software robots that perform Internet searching, looking for answers to questions, for related information, or for shopping deals. The searching techniques consider internet as a graph of nodes (pages) connected by links.

MEASURING PROBLEM-SOLVING PERFORMANCE

- The output of problem-solving algorithm is either failure or a solution. (Some algorithms might struck in an infinite loop and never return an output.)

- The algorithm's performance can be measured in four ways :
  - **Completeness**: Is the algorithm guaranteed to find a solution whenthere is one?
  - **Optinality** : Does the strategy find the optimal solution
  - **Time complexity:** How long does it take to find a solution?
  - **Space complexity:** How much memory is needed to perform thesearch?

UNINFORMED SEARCH STRATGES

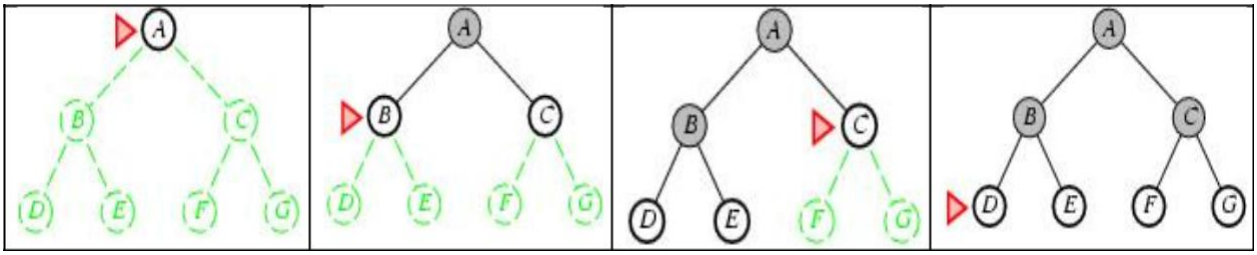
- **Uninformed Search Strategies** have no additional information about statesbeyond that provided in the **problem definition**.
- **Strategies** that know whether one non goal state is "more promising" thananother are called I**nformed search or heuristic search** strategies.

There are Six uninformed search strategies as given below.

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

Breadth-first search

- Breadth-first search is a simple strategy in which the root node is expanded first, then all successors of the root node are expanded next, then their successors, and so on.
  - In general, all the nodes are expanded at a given depth in the search tree before any nodes at the next level are expanded.
- Breath-first-search is implemented by calling TREE-SEARCH with an empty fringe that is a first-in-first-out(FIFO) queue, assuring that the nodes that are visited first will be expanded first.
- In otherwards, calling TREE-SEARCH (problem,FIFO-QVEVE()) results in breadth-first-search.
- The FIFO queue puts all newly generated successors at the end of the queue, which means that Shallow nodes are expanded before deeper nodes.



Breadth-first search on a simple binary tree. At each stage ,the node to be expanded next is indicated by a marker.

Properties of breadth-first-search

Complete?? Yes (if b is finite)

Time?? $1 + b + b^2 + b^3 + \ldots + b^d + b(b^d - 1) = O(b^{d+1})$, i.e., exp. in d

Space?? $O(b^{d+1})$ (keeps every node in memory)

Optimal?? No, unless step costs are constant

Space is the big problem; can easily generate nodes at 100MB/sec
so 24hrs = 8640GB.

Time and Memory Requirements for BFS – $O(b^{d+1})$

Example:

- b = 10
- 10000 nodes/second
- each node requires 1000 bytes of storage

Depth	Nodes	Time	Memory
2	1100	.11 sec	1 meg
4	111,100	11 sec	106 meg
6	10^7	19 min	10 gig
8	10^9	31 hrs	1 tera
10	10^{11}	129 days	101 tera
12	10^{13}	35 yrs	10 peta
14	10^{15}	3523 yrs	1 exa

Time complexity for BFS

- Assume every state has b successors.
- The root of the search tree generates b nodes at the first level,each of which generates b more nodes,for a total of b^2 at the second level.
- Each of these generates b more nodes,yielding b^3 nodes at the third level,and so on.
- Now suppose,that the solution is at depth d.
- In the worst case,we would expand all but the last node at level d,generating b^{d+1} - b nodes at level d+1.
- Then the total number of nodes generated is

  $b + b^2 + b^3 + \ldots + b^d + (bd^{+1} + b) = O(b^{d+1)}$.
- Every node that is generated must remain in memory,because it is either part of the fringe or is an ancestor of a fringe node.
- The space compleity is,therefore ,the same as the time complexity

UNIFORM-COST SEARCH

- Instead of expanding the shallowest node,**uniform-cost search** expands the node n with the lowest path cost.
- uniform-cost search does not care about the number of steps a path has,but only about their total cost.

Properties of Uniform-cost-search:

Expand least-cost unexpanded node

Implementation:
fringe = queue ordered by path cost, lowest first

Equivalent to breadth-first if step costs all equal

Complete?? Yes, if step cost $\geq \epsilon$

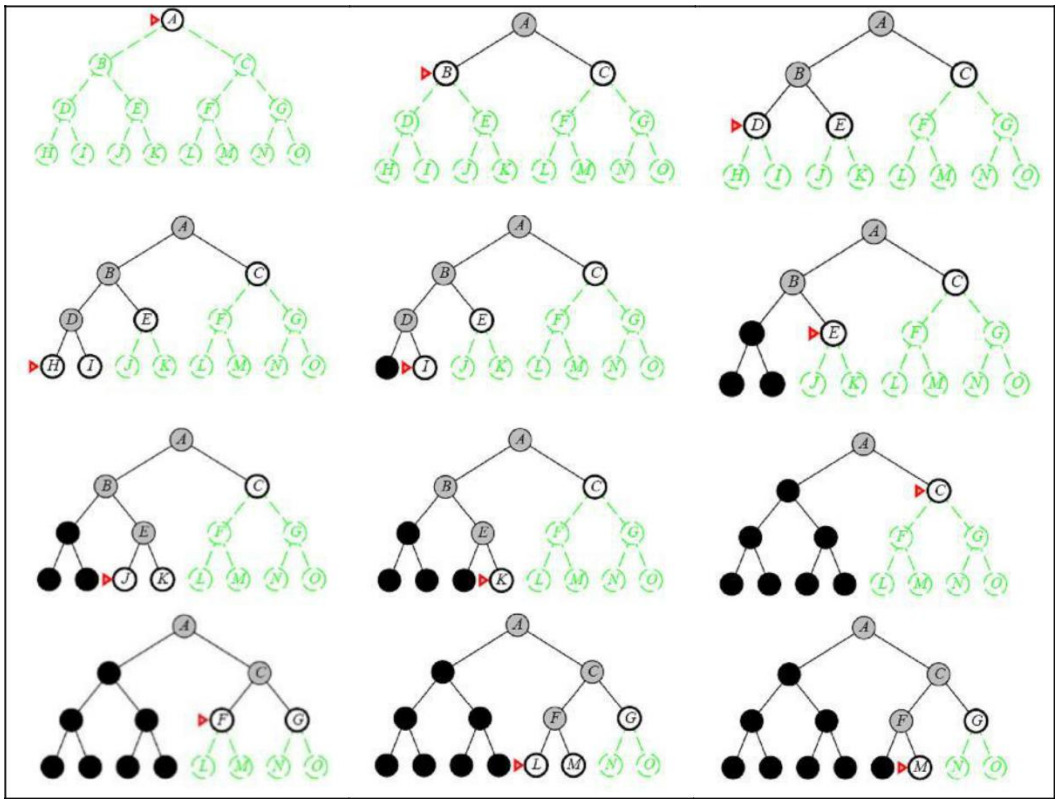
Time?? # of nodes with $g \leq$ cost of optimal solution, $O(b^{\lceil C^*/\epsilon \rceil})$
where C^* is the cost of the optimal solution

Space?? # of nodes with $g \leq$ cost of optimal solution, $O(b^{\lceil C^*/\epsilon \rceil})$

Optimal?? Yes—nodes expanded in increasing order of $g(n)$

DEPTH-FIRST-SEARCH

- Depth-first-search always expands the deepest node in the current fringe of the search tree.
- The progress of the search is illustrated in figure.
- The search proceeds immediately to the deepest level of the search tree,where the nodes have no successors.
- As those nodes are expanded,they are dropped from the fringe, so then the search "backs up" to the next shallowest node that still has unexplored successors.
- This strategy can be implemented by TREE-SEARCH with a last-in-first-out (LIFO) queue also known as a stack.
- Depth-first-search has very modest memory requirements.
- It needs to store only a single path from the root to a leaf node, along with the remaining unexpanded sibling nodes for each node on the path.
- Once the node has been expanded,it can be removed from the memory,as soon as its descendants have been fully explored.
- For a state space with a branching factor b and maximum depth m,depth-first-search requires storage of only bm + 1 nodes.



Depth-first-search on a binary tree. Nodes that have been expanded and have no descendants in the fringe can be removed from the memory; these are shown in black. Nodes at depth 3 are assumed to have no successors and M is the only goal node.

Drawback of Depth-first-search

The drawback of depth-first-search is that it can make a wrong choice and get stuck going down very long(or even infinite) path when a different choice would lead to solution near the root of the search tree.

For example, depth-first-search will explore the entire left subtree even if node C is a goal node.

BACKTRACKING SEARCH

A variant of depth-first search called backtracking search uses less memor y and only one successor is generated at a time rather than all successors.; Only O(m) memor y is needed rather than $O(b^m)$

DEPTH-LIMITED-SEARCH

- The problem of unbounded trees can be alleviated by supplying depth-first-search with a pre-determined depth limit l.
- That is, nodes at depth l are treated as if they have no successors. This approach is called **depth-limited-search**.
- The depth limit solves the infinite path problem.
- Depth limited search will be nonoptimal if we choose l > d. Its time complexity is $O(b^l)$ and its space compleiy is O(bl).
- Depth-first-search can be viewed as a special case of depth-limited search with l = oo Sometimes,depth limits can be based on knowledge of the problem.
- For,example,on the map of Romania there are 20 cities.
- Therefore,we know that if there is a solution.,it must be of length 19 at the longest,So l = 10 is a possible choice.
- However,it oocan be shown that any city can be reached from any other city in at most 9 steps.
- This number known as the **diameter** of the state space,gives us a better depth limit.
- Depth-limited-search can be implemented as a simple modification to the general tree-search algorithm or to the recursive depth-first-search algorithm.
- The pseudocode for recursive depth-limited-search is shown.

- It can be noted that the above algorithm can terminate with two kinds of failure : the standard *failure* value indicates no solution; the *cutoff* value indicates no solution within the depth limit.
- Depth-limited search = depth-first search with depth limit l, returns cut off if any path is cut off by depth limit

Recursive implementation of Depth-limited-search:

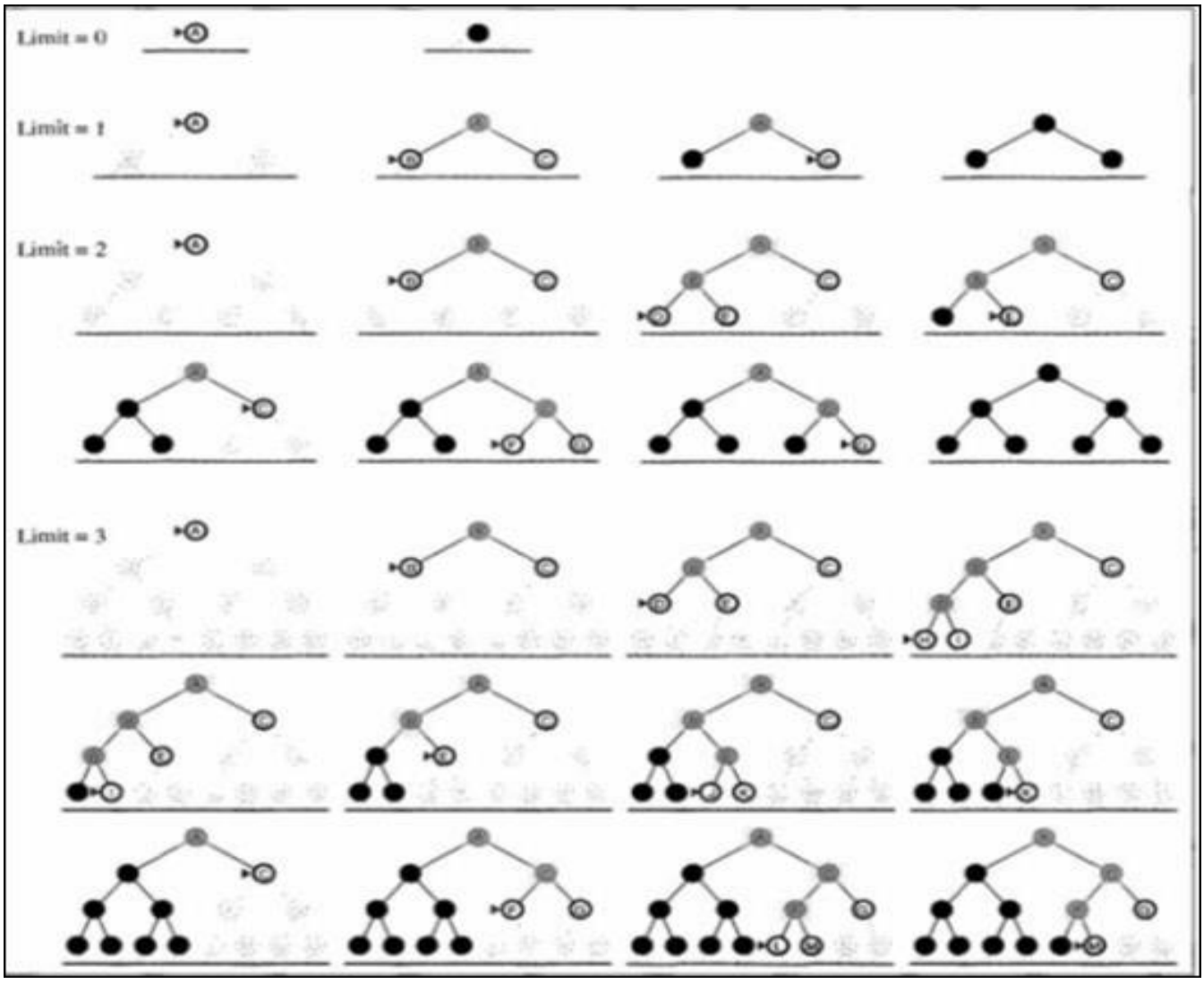
```
function Depth-Limited-Search( problem, limit) returns a solution/fail/cutoff
return Recursive-DLS(Make-Node(Initial-State[problem]), problem, limit)
function Recursive-DLS(node, problem, limit) returns solution/fail/cutoff
cutoff-occurred? ← false
if Goal-Test(problem,State[node]) then return Solution(node)
else if Depth[node] = limit then return cutoff
else for each successor in Expand(node, problem) do
result ← Recursive-DLS(successor, problem, limit)
if result = cutoff then cutoff_occurred? ← true
else if result not = failure then return result
if cutoff_occurred? then return cutoff else return failure
```

ITERATIVE DEEPENING DEPTH-FIRST SEARCH

- Iterative deepening search (or iterative-deepening-depth-first-search) is a general strategy often used in combination with depth-first-search,that finds the better depth limit.
- It does this by gradually increasing the limit - first 0,then 1,then 2, and so on - until a goal is found.
- This will occur when the depth limit reaches d,the depth of the shallowest goal node.
- Iterative deepening combines the benefits of depth-first and breadth-first-search Like depth-first-search, its memory requirements are modest; O(bd) to be precise.
- Like Breadth-first-search,it is complete when the branching factor is finite and optimal when the path cost is a non decreasing function of the depth of the node.
- The following figure shows the four iterations of ITERATIVE-DEEPENING_SEARCH on a binary search tree,where the solution is found on the fourth iteration.

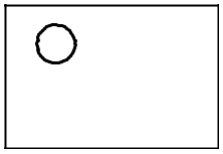
```
function ITERATIVE-DEEPENING-SEARCH( problem) returns a solution
    inputs: problem, a problem
    for depth ← 0 to ∞ do
        result ← DEPTH-LIMITED-SEARCH( problem, depth)
        if result ≠ cutoff then return result
    end
```

The **iterative deepening search algorithm**, which repeatedly applies depth-limited-search with increasing limits. It terminates when a solution is found or if the depth limited search returns *failure*, meaning that no solution exists.

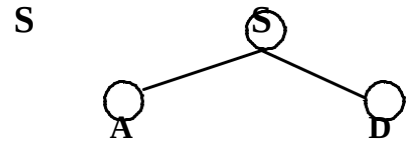


Four iterations of iterative deepening search on a binary tree

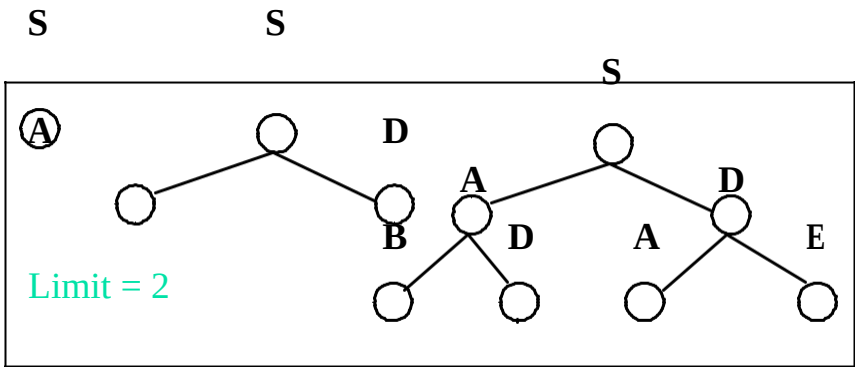
Iterative deepening search



Limit = 0



Limit = 1



Iterative search is not as wasteful as it might seem

Properties of iterative deepening search

Complete?? Yes

Time?? $(d+1)b^0 + db^1 + (d-1)b^2 + \ldots + b^d = O(b^d)$

Space?? $O(bd)$

Optimal?? No, unless step costs are constant
Can be modified to explore uniform-cost tree

Numerical comparison for $b = 10$ and $d = 5$, solution at far right leaf:

$$N(\text{IDS}) = 50 + 400 + 3,000 + 20,000 + 100,000 = 123,450$$
$$N(\text{BFS}) = 10 + 100 + 1,000 + 10,000 + 100,000 + 999,990 = 1,111,100$$

IDS does better because other nodes at depth d are not expanded

BFS can be modified to apply goal test when a node is generated

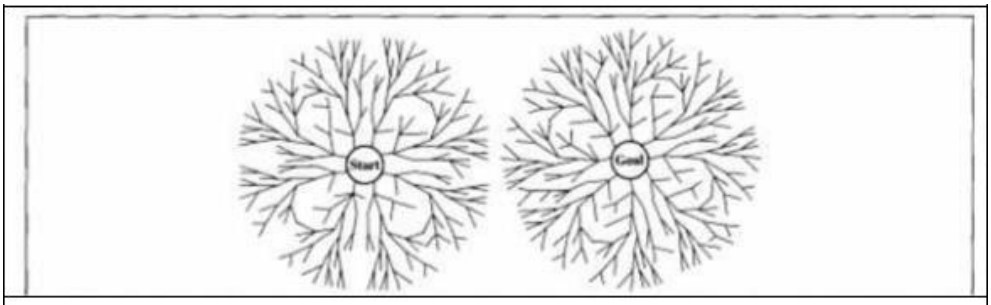
- In general,iterative deepening is the prefered uninformed search method when there is a large search space and the depth of solution is not known.

Bidirectional Search

The idea behind bidirectional search is to run two simultaneous searches

- one forward from the initial state and
- other backward from the goal,

It stops when the two searches meet in the middle. The motivation is that $b^{d/2} + b^{d/2}$ much less than $b^{d/}$



A schematic view of a bidirectional search that is about to succeed, when a Branch from the Start node meets a Branch from the goal node.

Comparing Uninformed Search Strategies

The following table compares search strategies in terms of the four evaluation criteria.

Criterion	Breadth-First	Uniform-Cost	Depth-First	Depth-Limited	Iterative Deepening	Bidirectional (if applicable)
Complete?	Yes[a]	Yes[a,b]	No	No	Yes[a]	Yes[a,d]
Time	$O(b^{d+1})$	$O(b^{1+\lfloor C^*/\epsilon \rfloor})$	$O(b^m)$	$O(b^\ell)$	$O(b^d)$	$O(b^{d/2})$
Space	$O(b^{d+1})$	$O(b^{1+\lfloor C^*/\epsilon \rfloor})$	$O(bm)$	$O(b\ell)$	$O(bd)$	$O(b^{d/2})$
Optimal?	Yes[c]	Yes	No	No	Yes[c]	Yes[c,d]

Evaluation of search strategies,b is the branching factor; d is the depth of the shallowest solution; m is the maximum depth of the search tree; l is the depth limit. Superscript caveats are as follows: [a] complete if b is finite; [b] complete if step costs >= E for positive E; [c] optimal if step costs are all identical; [d] if both directions use breadth-first search.

INFORMED SEARCH AND EXPLORATION

Informed (Heuristic) Search Strategies

- **Informed search strategy** is one that uses problem-specific knowledge beyond thedefinition of the problem itself.
- It can find solutions more efficiently than uninformed strategy.

Best-first search

- **Best-first search** is an instance of general TREE-SEARCH or GRAP H-SEARCHalgorithm in which a node is selected for expansion based on an **evaluation function** f(n).
- The node with lowest evaluation is selected for expansion,because the evaluation measures the distance to the goal.
- This can be implemented using a priority-queue,a data structure that will maintain the fringe in ascending order of f-values.

Heuristic functions

- A **heuristic function** or simply a **heuristic** is a function that ranks alternatives in various search algorithms at each branching step basing on an available information in order to make a decision which branch is to be followed during a search.
- The key component of Best-first search algorithm is a **heuristic function**,denoted by h(n):

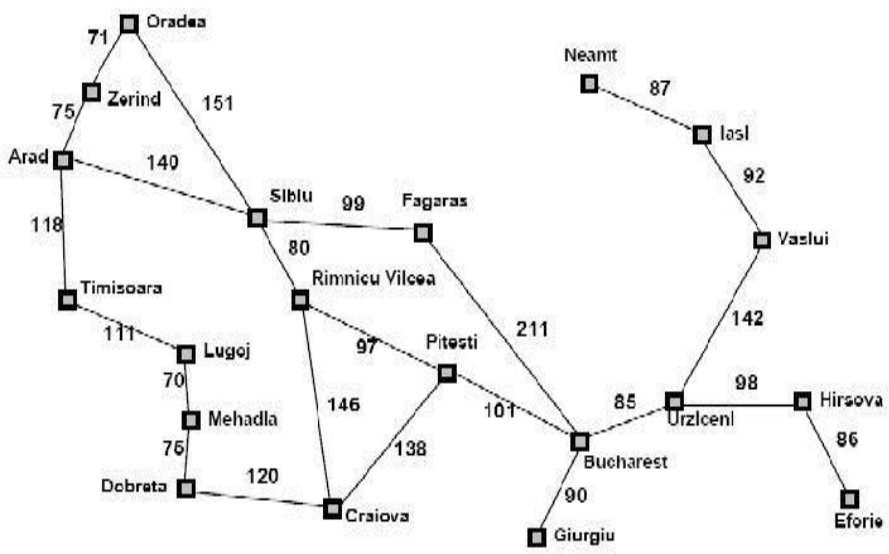
  h(n) = estimated cost of the **cheapest path** from node n to a **goal node**.
- For example,in Romania, one might estimate the cost of the cheapest path from Arad to Bucharest via a **straight-line distance** from Arad to Bucharest
- Heuristic function are the most common form in which additional knowledge is imparted to the search algorithm.

Greedy Best-first search

➔ **Greedy best-first search** tries to expand the node that is closest to the goal,on the grounds that this is likely to a solution quickly.
➔ It evaluates the nodes by using the heuristic function f(n) = h(n).
➔ Taking the example of **Route-finding problems** in Romania, the goal is to reach
➔ Bucharest starting from the city Arad.
➔ We need to know the straight-line distances to Bucharest from various cities.

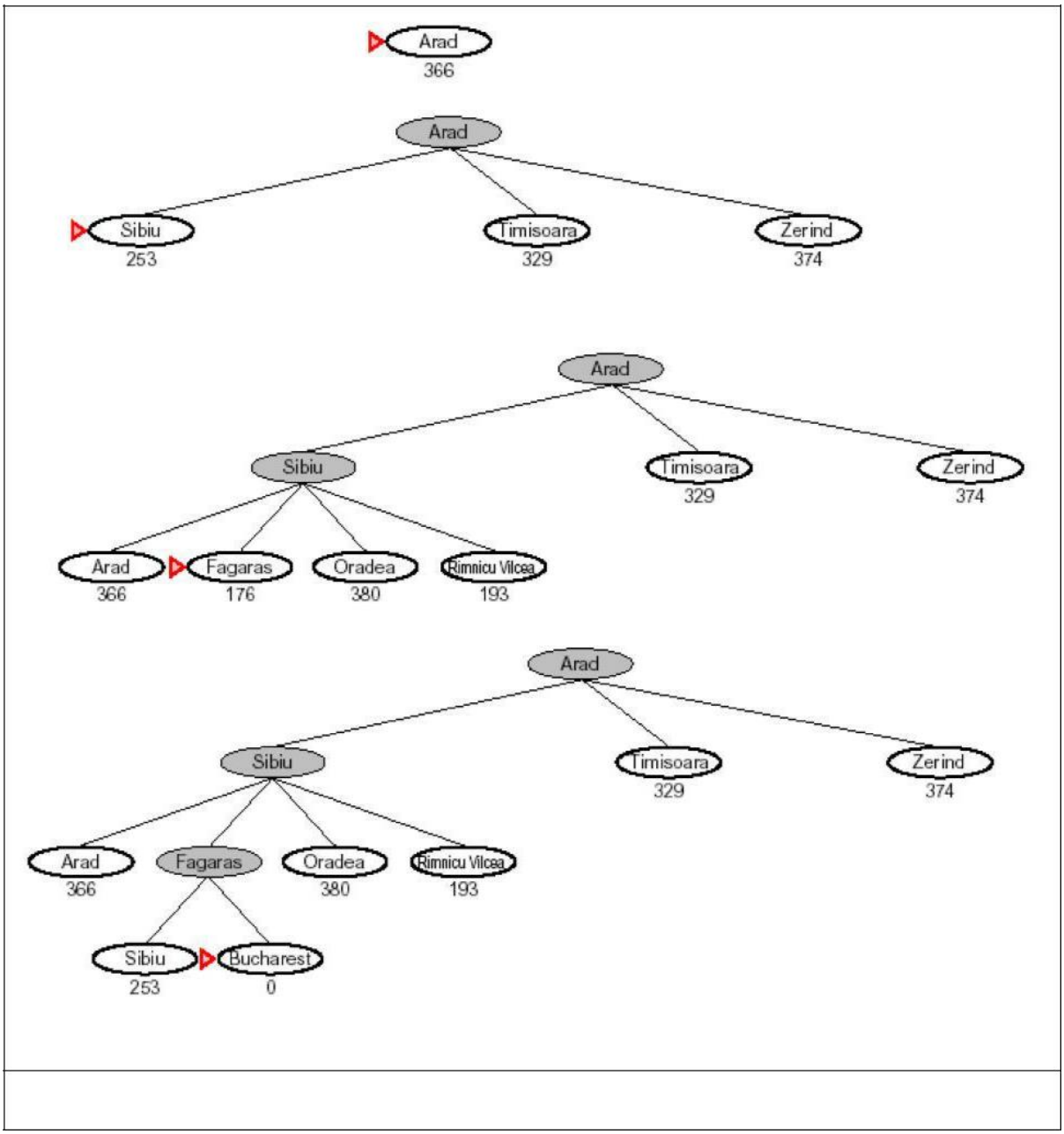
➔ For example, the initial state is In(Arad) ,and the straight line distance heuristic hSLD(In(Arad)) is found to be 366.

➔ Using the **straight-line distance** heuristic **hSLD ,**the goal state can be reached faster.



Straight line distance to Bucharest	
Arad	366
Bucharest	0
Craiova	160
Dobreta	242
Eforie	161
Fagaras	178
Giurgiu	77
Hirsova	151
Iasi	226
Lugoj	244
Mehadia	241
Neamt	234
Oradea	380
Pitesti	98
Rimnicu Vilcea	193
Sibiu	253
Timisoara	329
Urziceni	80
Vaslui	199
Zerind	374

Values of hSLD - straight line distances to Bucharest



Strategies in greedy best-first search for Bucharest using straight-line distance heuristic hSLD. Nodes are labeled with their h-values.

- The above figure shows the progress of greedy best-first search using hSLD 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.

Properties of greedy search

➔ **Complete??** No-can get stuck in loops, e.g.,
Iasi ! Neamt ! Iasi ! Neamt !
Complete in finite space with repeated-state checking

➔ **Time??** O(bm), but a good heuristic can give dramatic improvemento
➔ **Space??**O(bm)-keeps all nodes in memory
➔ **Optimal??**No

❖ Greedy best-first search is not optimal,and it is incomplete.

❖ The worst-case time and space complexity is $O(b^m)$,where m is the maximum depth of the search space.

A* Search

➔ **A* Search** is the most widely used form of best-first search. The evaluation functionf(n) is obtained by combining

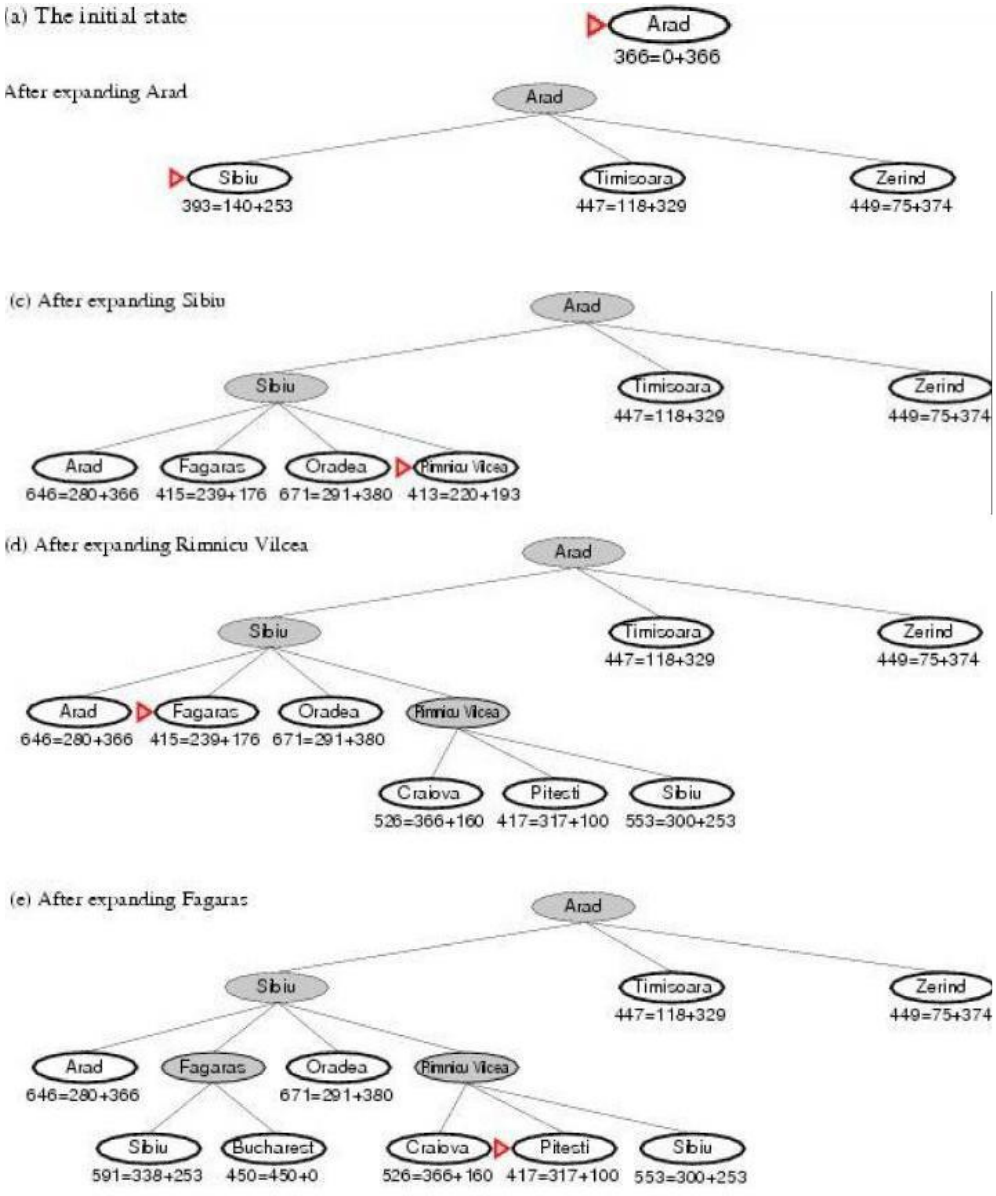
(1) **g(n) =** the cost to reach the node,and
(2) **h(n) =** the cost to get from the node to the **goal** :

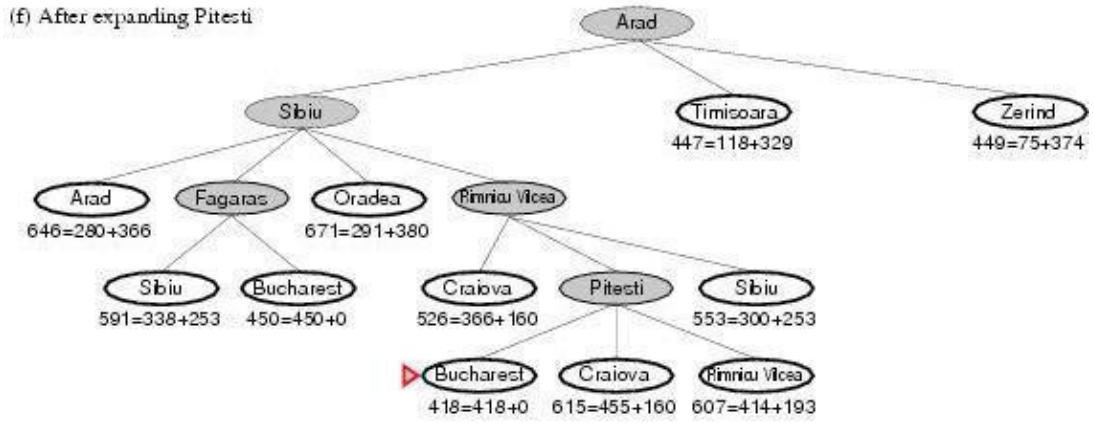
$$f(n) = g(n) + h(n).$$

➔ A* Search is both optimal and complete. AA* is optimal if h(n) is an admissible heuristic. The obvious example of admissible heuristic is the straight-line distance hSLD.
➔ It cannot be an overestimate.
➔ A*Search is optimal if h(n) is an admissible heuristic - that is, provided that h(n) never overestimates the cost to reach the goal.
➔ An obvious example of an admissible heuristic is the straight-line distance hSLD that we used in getting to Bucharest.
➔ The progress of an A* tree search for Bucharest is shown in above figure.
➔ The values of 'g ' are computed from the step costs shown in the Romania, Also the values of hSLD are given in Figure Route Map of Romania.

Recursive Best-first Search (RBFS)

- ➔ Recursive best-first search is a simple recursive algorithm that attempts to mimic the operation of standard best-first search, but using only linear space.
- ➔ The algorithm is shown in below figure.
- ➔ Its structure is similar to that of recursive depth-first search,but rather than continuing indefinitely down the current path,it keeps track of the f-value of the best alternative path available from any ancestor of the current node.
- ➔ If the current node exceeds this limit, the recursion unwinds back to the alternative path. As the recursion unwinds, RBFS replaces the f-value of each node along the path with the best f-value of its children.



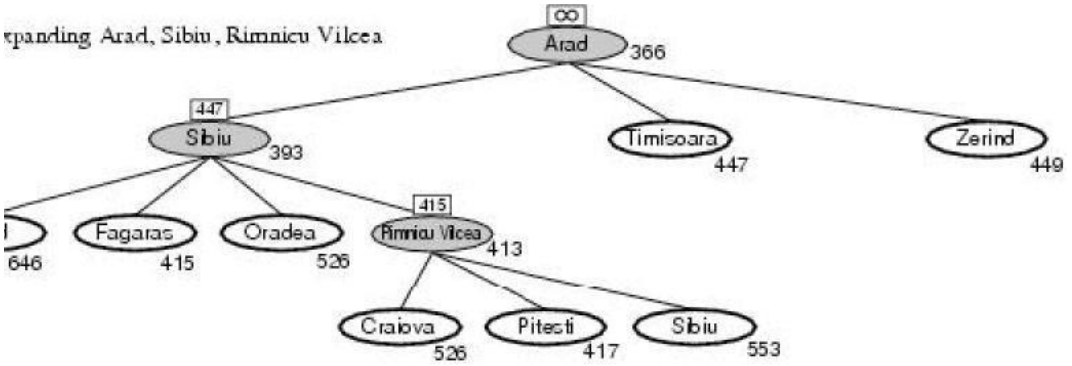


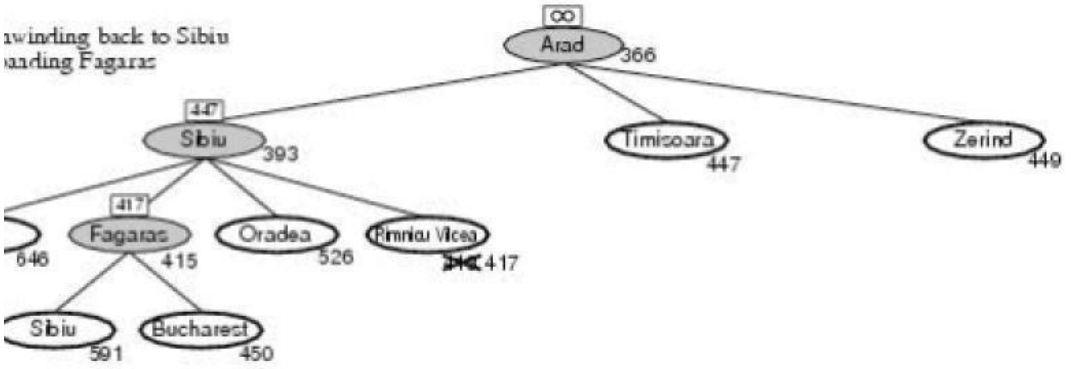
Stages in A* Search for Bucharest. Nodes are labeled with f = g + h . The h-values are the straight-line distances to Bucharest taken from figure Route map of

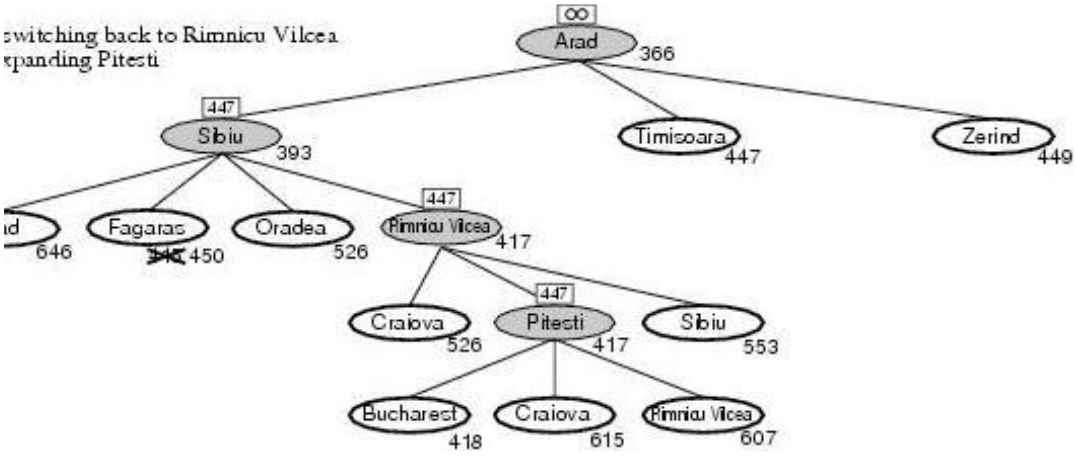
```
ction RECURSIVE-BEST-FIRST-SEARCH(problem) return a solution or failure
eturn RFBS(problem,MAKE-NODE(INITIAL-STATE[problem]),∞)

tion RFBS( problem, node, f_limit) return a solution or failure and a new f-
ost limit
f GOAL-TEST[problem](STATE[node]) then return node
uccessors ← EXPAND(node, problem)
f successors is empty then return failure, ∞
or each s in successors do
    f [s] — max(g(s) + h(s), f [node])
epeat
    best — the lowest f-value node in successors
    if f [best] > f_limit then return failure, f [best]
    alternative — the second lowest f-value among successors
    result, f [best] — RBFS(problem, best, min(f_limit, alternative))
    if result ≠ failure then return result
```

The algorithm for recursive best-first search


xpanding Arad, Sibiu, Rimnicu Vilcea
∞
Arad
366
447
Sibiu
393
Timisoara
447
Zerind
449
646
Fagaras
415
Oradea
526
415
Rimnicu Vilcea
413
Craiova
526
Pitesti
417
Sibiu
553


winding back to Sibiu
anding Fagaras
∞
Arad
366
447
Sibiu
393
Timisoara
447
Zerind
449
417
Fagaras
415
646
Oradea
526
Rimnicu Vilcea
417
Sibiu
591
Bucharest
450


witching back to Rimnicu Vilcea
xpanding Pitesti
∞
Arad
366
447
Sibiu
393
Timisoara
447
Zerind
449
646
Fagaras
450
Oradea
526
447
Rimnicu Vilcea
417
447
Craiova
526
Pitesti
417
Sibiu
553
Bucharest
418
Craiova
615
Rimnicu Vilcea
607

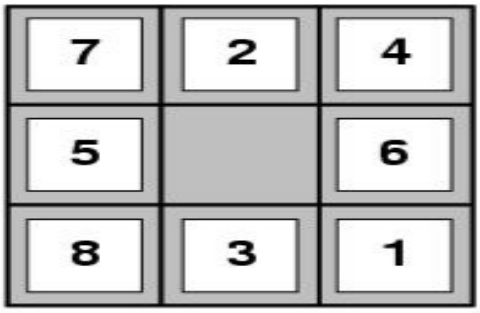
- Stages in an RBFS search for the shortest route to Bucharest. The f-limit value for each recursive call is shown on top of each current node.
- (a) The path via Rimnicu Vilcea is followed until the current best leaf (Pitesti) has a value that is worse than the best alternative path (Fagaras).
- (b) The recursion unwinds and the best leaf value of the forgotten subtree (417) is backed up to Rimnicu Vilcea;then Fagaras is expanded,revealing a best leaf value of 450.
- (c) The recursion unwinds and the best leaf value of the forgotten subtree (450) is backed upto Fagaras; then Rimni Vicea is expanded.
- This time because the best alternative path(through Timisoara) costs atleast 447,the expansion continues to Bucharest

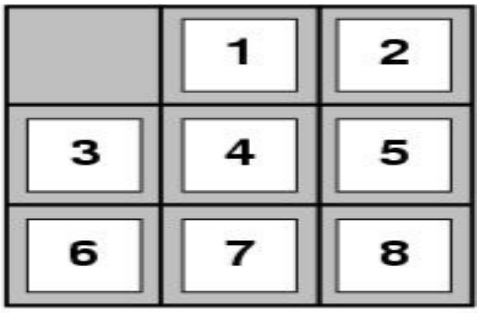
RBFS Evaluation:

- RBFS is a bit more efficient than IDA*
  - Still excessive node generation (mind changes)
- Like A*, optimal if *h(n)* is admissible
- Space complexity is *O(bd)*.
  - IDA* retains only one single number (the current f-cost limit)
- Time complexity difficult to characterize
  - Depends on accuracy if h(n) and how often best path changes.
- IDA* en RBFS suffer from ***too little*** memory.

Heuristic Functions

A **heuristic function** or simply a heuristic is a function that ranks alternatives in various search algorithms at each branching step basing on an available information in order to make a decision which branch is to be followed during a search





A typical instance of 8 – puzzle

-puzzle is an example of Heuristic search problem.

- The object of the puzzle is to slide the tiles horizontally or vertically into the empty space until the configuration matches the goal configuration
- The average cost for a randomly generated 8-puzzle instance is about 22 steps.
- The branching factor is about 3.(When the empty tile is in the middle,there are four possible moves;when it is in the corner there are two;and when it is along an edge there are three).

The solution is 26 steps long.

- This means that an exhaustive search to depth 22 would look at about 3^{22} approximately = 3.1 X 10^{10} states.
- By keeping track of repeated states, we could cut this down by a factor of about 170, 000, because there are only 9!/2 = 181,440 distinct states that are reachable.
- This is a manageable number, but the corresponding number for the 15-puzzle is roughly 10^{13}.
- If we want to find the shortest solutions by using A^*,we need a heuristic function that never overestimates the number of steps to the goal.
- The two commonly used heuristic functions for the 15-puzzle are :
    - h_1 = the number of misplaced tiles.
- In the above figure all of the eight tiles are out of position, so the start state would have $h_1 = 8$. h_1 is an admissible heuristic.
    - h_2 = the sum of the distances of the tiles from their goal positions. This is called **the city block distance** or **Manhattan distance.**
- h_2 is admissible ,because all any move can do is move one tile one step closer to the Tiles 1 to 8 in start state give a Manhattan distance of
    - $h_2 = 3 + 1 + 2 + 2 + 2 + 3 + 3 + 2 = 18$.
- Neither of these overestimates the true solution cost, which is 26.
- The

The 8

The Effective Branching factor

- One way to characterize the **quality of a heuristic** is the **effective branching factor b*.** If the total number of nodes generated by A* for a particular problem is **N**,and the **solution depth** is **d,**then $b^*$is the branching factor that a uniform tree of depth d wouldhave to have in order to contain N+1 nodes. Thus,
  - $N + 1 = 1 + b^* + (b^*)^2 + \ldots + (b^*)^d$
- For example,if A^* finds a solution at depth 5 using 52 nodes,then effective branching factor is 1.92.
- A well designed heuristic would have a value of b^* close to 1,allowing failru large problems to be solved.
- To test the heuristic functions h1 and h2,1200 random problems were generated with solution lengths from 2 to 24 and solved them with iterative deepening search and with A^* search using both h1 and h2.
- The following table gives the average number of nodes expanded by each strategy and the effective branching factor.
- The results suggest that h_2 is better than h_1,and is far better than using iterative deepening search.
- For a solution length of 14,A^* with h2 is 30,000 times more efficient than uninformed iterative deepening search.

	Search Cost			Effective Branching Factor		
d	IDS	$A^*(h_1)$	$A^*(h_2)$	IDS	$A^*(h_1)$	$A^*(h_2)$
2	10	6	6	2.45	1.79	1.79
4	112	13	12	2.87	1.48	1.45
6	680	20	18	2.73	1.34	1.30
8	6384	39	25	2.80	1.33	1.24
10	47127	93	39	2.79	1.38	1.22
12	3644035	227	73	2.78	1.42	1.24
14	–	539	113	–	1.44	1.23
16	–	1301	211	–	1.45	1.25
18	–	3056	363	–	1.46	1.26
20	–	7276	676	–	1.47	1.27
22	–	18094	1219	–	1.48	1.28
24	–	39135	1641	–	1.48	1.26

Comparison of search costs and effective branching factors for the ITERATIVE-DEEPENING- SEARCH and A^* Algorithms with h_1,and h_2. Data are average over 100 instances of the 8-puzzle,for various solution lengths.

Inventing admissible heuristic functions

Relaxed problems

- A problem with fewer restrictions on the actions is called a ***relaxed problem***
- The cost of an optimal solution to a relaxed problem is an admissible heuristic for the original problem
- If the rules of the 8-puzzle are relaxed so that a tile can move *anywhere*, then *hi(n)* gives the shortest solution
- If the rules are relaxed so that a tile can move to *any adjacent square,* then *h2(n)* gives the shortest solution

CONSTRAINT SATISFACTION PROBLEMS (CSP)

A **Constraint Satisfaction Problem** (or CSP) is defined by a

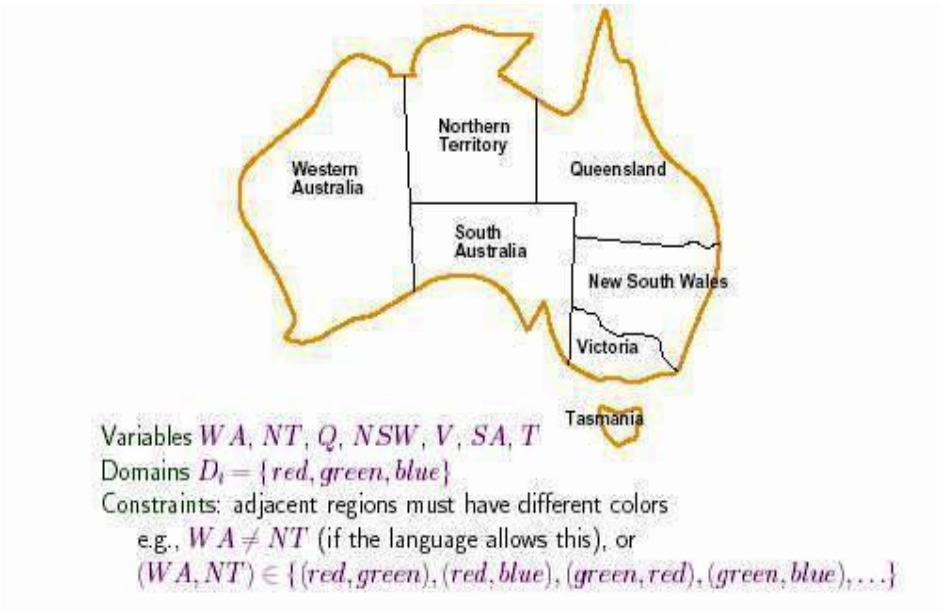
- set of **variables** X_1, X2….X_n, and a
- set of constraints C1,C2,…,Cm.
- Each variable X_i has a nonempty **domain** D,of possible **values**.
- Each constraint Ci involves some subset of variables and specifies the allowable combinations of values for that subset.

- A **State** of the problem is defined by an **assignment** of values to some or all of the variables,{Xi = vi,Xj = vj,…}.
- An assignment that does not violate any constraints is called a **consistent** or **legalassignment.**
- A complete assignment is one in which every variable is mentioned, and a **solution** to a CSP is a complete assignment that satisfies all the constraints.
- Some CSPs also require a solution that maximizes an **objective function**. For Example for Constraint Satisfaction Problem :

- The following figure shows the map of Australia showing each of its states and territories.
  - We are given the task of coloring each region either red,green,or blue in such a way that the neighboring regions have the same color.

- To formulate this as CSP ,we define the variable to be the regions :WA,NT,Q,NSW,V,SA, and T.

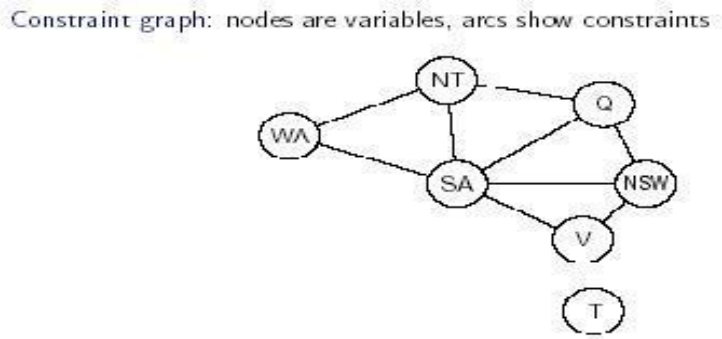
- The domain of each variable is the set {red,green,blue}.
- The constraints require neighboring regions to have distinct colors;
  - for example, the allowable combinations for WA and NT are the pairs {(red,green),(red,blue),(green,red),(green,blue),(blue,red),(blue,green)}.
  - The constraint can also be represented more succinctly as the inequality WA not = NT,provided the constraint satisfaction algorithm has some way to evaluate such expressions.)
- There are many possible solutions such as
  - { WA = red, NT = green,Q = red, NSW = green, V = red ,SA = blue,T = red}.



Principle states and territories of Australia. Coloring this map can be viewed as aconstraint satisfaction problem. The goal is to assign colors to each region so that no neighboring regions have the same color.

- It is helpful to visualize a CSP as a constraint graph,as shown in the following figure.

- The nodes of the graph corresponds to variables of the problem and the arcs correspond to constraints.



The map coloring problem represented as a constraint graph.

CSP can be viewed as a standard search problem as follows :

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

Every solution must be a complete assignment and therefore appears at depth n if there are n variables.

Depth first search algorithms are popular for CSPs

Varieties of CSPs

(i) Discrete variables

Finite domains

- The simplest kind of CSP involves variables that are **discrete** and have **finite domains.** Map coloring problems are of this kind.
- The 8-queens problem can also be viewed as finite-domain
- CSP,where the variables Q_1,Q_2,…..Q_8 are the positions each queen in columns 1,….8 and each variable has the domain {1,2,3,4,5,6,7,8}.

- If the maximum domain size of any variable in a CSP is d,then the number of possible complete assignments is $O(d^n)$ - that is,exponential in the number of variables.
- Finite domain CSPs include **Boolean CSPs**,whose variables can be either *true* or *false*.

Infinite domains

- Discrete variables can also have **infinite domains** - for example,the set of integers or the set of strings.
- With infinite domains, it is no longer possible to describe constraints by enumerating all allowed combination of values. Instead a constraint language of algebric inequalities such as Startjob1 + 5 <= Startjob3.

(ii) CSPs with continuous domains

- CSPs with continuous domains are very common in real world.
- For example ,in operation research field,the scheduling of experiments on the Hubble
- Telescope requires very precise timing of observations; the start and finish of each observation and maneuver are continuous-valued variables that must obey a variety of astronomical,precedence and power constraints.
- The best known category of continuous-domain CSPs is that of **linear programming** problems,where the constraints must be linear inequalities forming a *convex* region.
- Linear programming problems can be solved in time polynomial in the number of variables.

Varieties of constraints :

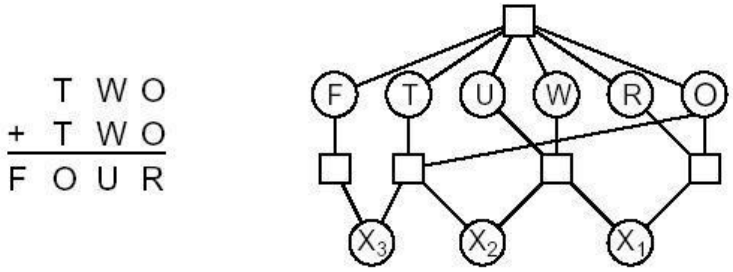
(i)Unary constraints involve a single variable.
Example: SA # green

(ii) **Binary constraints** involve pairs of variables.
Example: SA # WA

(iii) **Higher order constraints** involve 3 or more variables.
Example: cryptarithmetic puzzles.

(iv) **Absolute constraints** are the constraints, which rules out a potential solution when they are violated

(v)**Preference constraints** are the constraints indicating which solutions are preferred
Example: Vniversity Time Tabling Problem



Variables: $F\ T\ U\ W\ R\ O\ X_1\ X_2\ X_3$
Domains: $\{0,1,2,3,4,5,6,7,8,9\}$
Constraints
alldiff(F,T,U,W,R,O)
$O+O=R+10\cdot X_1$, etc.

- Cryptarithmetic problem.
- Each letter stands for a distinct digit;the aim is to find a substitution of digits for letters such that the resulting sum is arithmetically correct,with the added restriction that no leading zeros are allowed.
- The constraint hypergraph for the cryptarithmetic problem,showint the *Alldiff* constraint as well as the column addition constraints.
- Each constraint is a square box connected to the variables it contains.

Backtracking Search for CSPs

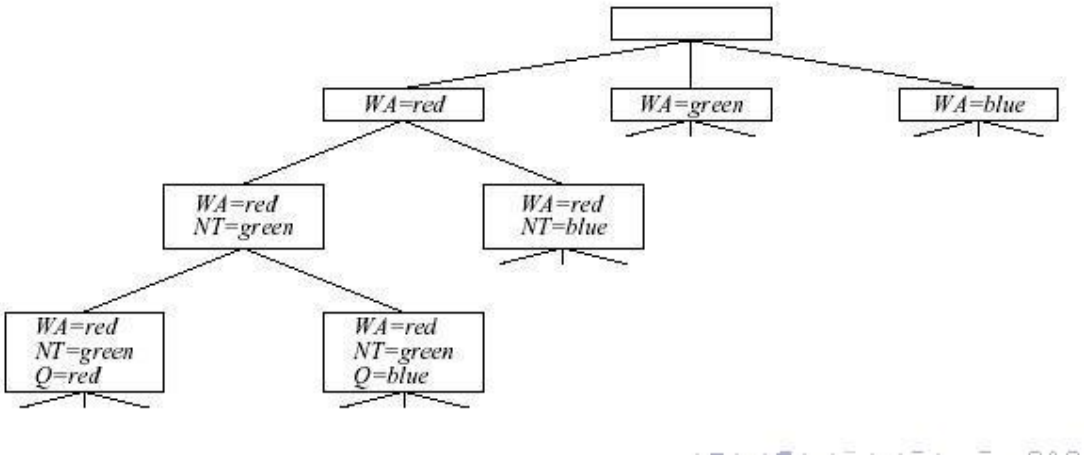
- The term **backtracking search** is used for depth-first search that chooses values for one variable at a time and backtracks when a variable has no legal values left to assign.

- The following algorithm shows the Backtracking Search for CSP

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

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

A simple backtracking algorithm for constraint satisfaction problem. The algorithm is modeled on the recursive depth-first search



Part of search tree generated by simple backtracking for the map coloring problem.

Propagating information through constraints

- So far our search algorithm considers the constraints on a variable only at the time that the Variable is chosen by SELECT-VNASSIGNED-VARIABLE.
- But by looking at some of the constraints earlier in the search, or even before the search has started, we can drastically reduce the search space.

Forward checking

- One way to make better use of constraints during search is called **forward checking**.
- Whenever a variable X is assigned, the forward checking process looks at each unas signed variable Y that is connected to X by a constraint and deletes from Y 's domain any value that is inconsistent with the value chosen for X.
- The following figure shows the progress of a map-coloring search with forward checking.

	WA	*NT*	*Q*	*NSW*	*V*	*SA*	*T*
Initial domains	R G B	R G B	R G B	R G B	R G B	R G B	R G B
After *WA*=*red*	(R)	G B	R G B	R G B	R G B	G B	R G B
After *Q*=*green*	(R)	B	(G)	R B	R G B	B	R G B
After *V*=*blue*	(R)	B	(G)	R	(B)		R G B

Figure 5.6 The progress of a map-coloring search with forward checking. $WA = red$ is assigned first; then forward checking deletes *red* from the domains of the neighboring variables NT and SA. After $Q = green$, *green* is deleted from the domains of NT, SA, and NSW. After $V = blue$, *blue* is deleted from the domains of NSW and SA, leaving SA with no legal values.

Constraint propagation

Although forward checking detects many inconsistencies, it does not detect all of them.

Constraint propagation is the general term for propagating the implications of a constrainton one variable onto other variables.

Arc Consistency

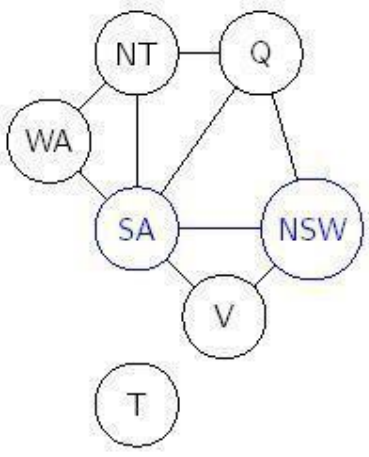


Figure: Australian Territories

- One method of constraint propagation is to enforce **arc consistency**
  - Stronger than forward checking
  - Fast
- *Arc* refers to a *directed* arc in the constraint graph
- Consider two nodes in the constraint graph (e.g., *SA* and *NSW*)
  - An arc is **consistent** if
  - For every value x of *SA*
  - There is some value y of *NSW* that is consistent with x
- Examine arcs for consistency in *both* directions

K-Consistency

- Can define stronger forms of consistency

k-Consistency

A CSP is k-**consistent** if, for any consistent assignment to $k-1$ variables, there is a consistent assignment for the k-th variable

- **1-consistency (node consistency)**
  - Each variable by itself is consistent (has a non-empty domain)
- **2-consistency (arc consistency)**
- **3-consistency (path consistency)**
  - Any pair of adjacent variables can be extended to a third

Local Search for CSPs

- Local search algorithms good for many CSPs
- Use complete-state formulation
  - Value assigned to every variable
  - Successor function changes one value at a time
- Have already seen this
  - Hill climbing for 8-queens problem (AIMA § 4.3)
- Choose values using **min-conflicts** heuristic
  - Value that results in the minimum number of conflicts with other variables

The Structure of Problems

Problem Structure

- Consider ways in which the structure of the problem's constraint graph can help find solutions
- Real-world problems require decomposition into subproblems

Independent Sub problems

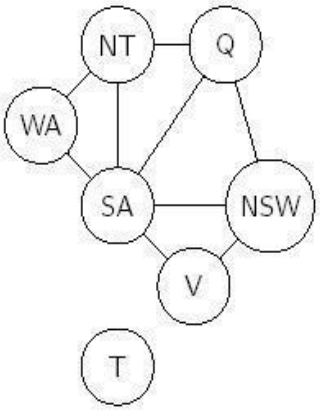


Figure: Australian Territories

- T is not connected
- Coloring T and coloring remaining nodes are **independent subproblems**
- *Any* solution for T combined with *any* solution for remaining nodes solves the problem
- Independent subproblems correspond to **connected components** of the constraint graph
- Sadly, such problems are rare

Tree-Structured CSPs

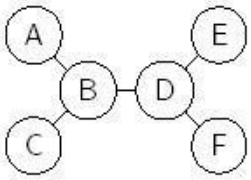


Figure: Tree-Structured CSP

- In most cases, CSPs are connected
- A simple case is when the constraint graph is a **tree**
- Can be solved in time linear in the number of variables
  - Order variables so that each parent precedes its children
  - Working "backward," apply arc consistency between child and parent
  - Working "forward," assign values consistent with parent

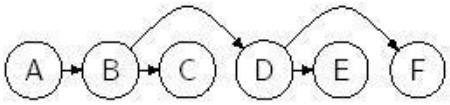


Figure: Linear ordering

UNIT II

REASONING

Symbolic Reasoning Under Uncertainty- Statistical Reasoning - Weak Slot-And-Filler-Structure - Semantic nets – Frames- Strong Slot-And-Filler Structure-Conceptual Dependency-Scripts- CYC.

What is Knowledge Representation in AI?

In the field of AI, there are many complex tasks required to evaluate either in the field of Machine learning or deep learning. It is indeed necessary to automate a knowledge processing system in such system. Knowledge representation is one such process which depends on the logical situation and enable a strategy to take a decision in acquiring knowledge. There are many types and levels of knowledge acquired by human in daily life but machines find difficult to interpret all types of knowledge. For such conditions, knowledge representation is used.

In knowledge representation algorithms, AI agents tend to think and they contribute in taking decisions. With the aid of such complex thinking, they are capable to solve the complex problems indulged in real world scenarios that are hard and time consuming for a human being to interpret.

Example Systems

We will take a look at three implemented systems

- Cognitive Assistant (SIRI)
- Smart Textbook (Inquire)
- Computational Knowledge Engine (Wolfram Alpha)

For each system, we will look at

- What knowledge must it represent?
- What reasoning must it do?
- What would it take to extend it?
- Where does it fail?
- How is it different from (current) Google?

Cognitive Assistant SIRI

See Demo at: http://www.youtube.com/watch?v=MpjpVAB06O4&feature=player_embedded

• What knowledge must it represent?
  - Restaurants, movies, events, reviews, …
  - Location, tasks, web sources, …

• What reasoning must it do?
  - Nearest location, date for tomorrow, AM vs PM, etc

• What would it take to extend it?
  – More sources, different sources,
• Where does it fail?
  – Completely different environment, completely different task
• Differences from Google
  – Dialog driven, task-oriented, location aware, …

Smart Textbook Inquire

What knowledge must it represent? (Demo in the class)
  – Concepts, definitions, relationships, descriptions
• What reasoning must it do?
  – Follow relationships, answer questions
• What would it take to extend it?
  – Must be customized to a new domain, must have methods for handling each kind ofquestion
• Where does it fail?
  – Does not capture all the content in the book, limited forms of reasoning
• How is it different from Google?
  – Very specific domain targeted at a specific class of user situated in an educational context

Wolfram Alpha

Try out examples at: http://www.wolframalpha.com/examples/
  – We will focus on the nutrition example
• What knowledge must it represent?
  – Different kinds of foods, their nutrition composition, caloric values
• What reasoning must it do?
  – Mathematical computations based on portions
• What would it take to extend it?
  – Add more data on foods and nutrition composition
• Where does it fail?
  – Does not know about recipes, how to combine foods, …
• How is it different from Google?
  – Data driven as opposed to document driven, mathematical reasoning

What do you mean by the term "Reasoning"?

Reasoning is an act of deriving a conclusion from certain premises using a given methodology.

- Reasoning is a process of thinking; logically arguing; drawing inference.
- When a system is required to do something, that it has not been explicitly told how to do, it must reason. It must figure out what it needs to know from what it already knows.
- Many types of reasoning have been identified and recognized, but many questions regarding their logical and computational properties still remained under controversy.
- Popular methods of Reasoning include: Abduction, Induction, model- based, explanation and confirmation. All of them are intimately related to problems of belief revision and theory development, knowledge assimilation, discovery and learning.
- When a system is required to do something, that it has not been explicitly told how to do, it must reason. It must figure out what it needs to know from what it already knows.

Example:1

Fact-1 : Robins are birds, Fact-2 : All birds have wings.

Then we can ask: DO ROBINS HAVE WINGS?

Hence to answer the above question- some reasoning must go

Example:2

Given

– Patient X allergic to medication M

– Anyone allergic to medication M is also allergic to medication M'

Reasoning helps us derive

– Patient X is allergic to medication M'

Human reasoning capabilities

- Broadly it is being divided into three areas:
  - Mathematical Reasoning- *axioms, definitions, theorems, proofs*
  - Logical Reasoning- *deductive, inductive, abductive*
  - Non-logical Reasoning- linguistic, language
- Above three mentioned are in every human being, but the ability level depends on education, genetics and environment.
- Intelligent Quotient (IQ)= mathematical + logical reasoning, whereas, Emotional Quotient mostly depends on non-logical reasoning capabilities.
- Logical Reasoning is our major concern in AI.

Logical Reasoning

➔ **Logic** is a language of reasoning. It is a collection of rules called logic arguments, we use when doing logical reasoning.

- ➔ **Logical Reasoning** is a process of drawing conclusions from premises using rule of inference.
- ➔ The study of is divided into two: **formal and informal logic.**
- ➔ The formal logic is sometimes called symbolic logic.
- ➔ **Symbolic logic** is the study of symbolic abstraction (construct) that capture the formal features of logical inference by a formal system.
- ➔ **Formal system** consists of two components, a formal language and a set of inference rules. The formal system has axioms.
- ➔ **Axiom** is a sentence that is always true within the system.
- ➔ **Sentences** are derived using the system's axioms and rules of derivation are called theorems.

Formal Logic

- The formal logic is the study of inference with purely formal content, i.e. where content is made explicit.

  Eg: Propositional logic and Predicate Logic.

- Here the logical arguments are set of rules for manipulating symbols. The rules are of two types:
  - Syntax rules: how to build meaningful expressions.
  - Inference rules: how to obtain true formulas from other true formulas.

- Logic also need **semantics,** which says how to assign meaning to expressions.

Informal Logic

- The informal logic is study of natural language arguments.
- The analysis of argument structures in ordinary language is part of informal logic
- The focus lies in distinguishing good arguments (valid) and bad arguments (invalid).

Formal systems

- Formal system can have following three properties:
  - **Consistency:** system's theorems do not contradict
  - **Soundness:** system's rules of derivation will never infer anything false, so long as start is with only true premises.
- **Completeness:** there are no true sentences in the system that cannot be proved using derivation rules of the system.
- **Elements of formal systems:**
  - The finite set of symbols for constructing formulae.
  - A grammar, is the way of constructing well-formed formulae (wff).
  - A set of axioms; each axiom has to be a wff.
  - A set of inference rules.
  - A set of theorems.

- A well-formed formulae, wff, is any string generated by a grammar.

  Example: the sequence of symbols ((aàb)à(~bà~a)) is a WFF because its is grammatically correct in propositional logic.

Formal Language

- A formal language may be viewed as being analogous to a collection of words or a collection of sentences.
- In computer science, a formal language is defined as precise mathematical or machine process able formulas.
- A formal language L is characterized as a set F of finite length sequences of elements drawn from a specified finite set A of symbols.
- The mathematical theory that treats formal language in general is known as **formal language theory.**

Uncertainty in Reasoning

- The world is an uncertain places; often the knowledge is imperfect which causes uncertainty. Therefore reasoning must be able to operate under uncertainty.
- AI systems must have ability to reason under conditions of uncertainty.
  - Incompleteness knowledge - compensate for lack of knowledge.
  - Inconsistencies knowledge - Resolve ambiguities and contradictions
  - Changing knowledge - update the knowledgebase over time.

Methods of reasoning

To a certain extend this will depend on the chosen knowledge representation Although a good knowledge representation scheme has to allow easy, natural, and plausible reasoning.

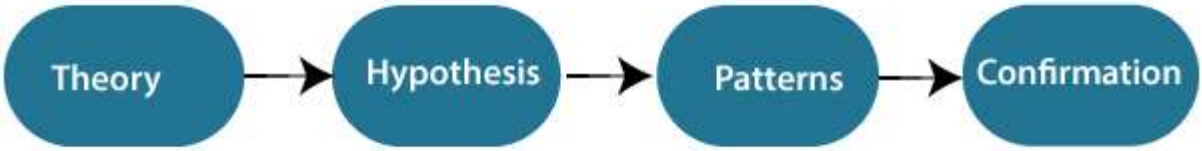
- Three types of logical reasoning:
  - **Deduction**
  - **Induction**
  - **Abduction**
  - Common Sense Reasoning
  - Monotonic Reasoning
  - Non-monotonic Reasoning

1. Deductive reasoning:

- Deductive reasoning is deducing new information from logically related known information. It is the form of valid reasoning, which means the argument's conclusion must be true when the premises are true.
- Deductive reasoning is a type of propositional logic in AI, and it requires various rules and facts. It is sometimes referred to as top-down reasoning, and contradictory to inductive reasoning.

- In deductive reasoning, the truth of the premises guarantees the truth of the conclusion.
- Deductive reasoning mostly starts from the general premises to the specific conclusion, which can be explained as below example.

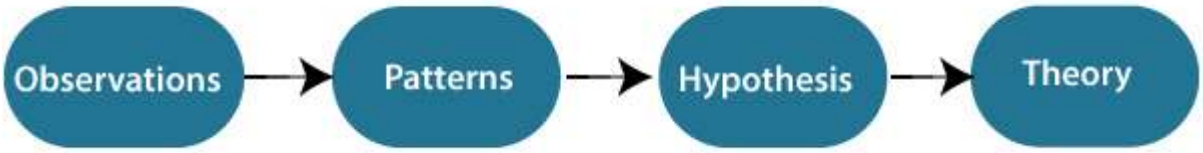
Example:
Premise-1: All the human eats veggies
Premise-2: Suresh is human.
Conclusion: Suresh eats veggies.
The general process of deductive reasoning is given below:



2. Inductive Reasoning:

- Inductive reasoning is a form of reasoning to arrive at a conclusion using limited sets of facts by the process of generalization. It starts with the series of specific facts or data and reaches to a general statement or conclusion.
- Inductive reasoning is a type of propositional logic, which is also known as cause-effect reasoning or bottom-up reasoning.
- In inductive reasoning, we use historical data or various premises to generate a generic rule, for which premises support the conclusion.
- In inductive reasoning, premises provide probable supports to the conclusion, so the truth of premises does not guarantee the truth of the conclusion.

Example:
Premise: All of the pigeons we have seen in the zoo are white.
Conclusion: Therefore, we can expect all the pigeons to be white.



3.Abductive reasoning:

Abductive reasoning is a form of logical reasoning which starts with single or multiple observations then seeks to find the most likely explanation or conclusion for the observation.
Abductive reasoning is an extension of deductive reasoning, but in abductive reasoning, the premises do not guarantee the conclusion.
Example:
Implication: Cricket ground is wet if it is raining
Axiom: Cricket ground is wet.
Conclusion It is raining.

4. Common Sense Reasoning

Common sense reasoning is an informal form of reasoning, which can be gained through experiences. Common Sense reasoning simulates the human ability to make presumptions about events which occurs on every day. It relies on good judgment rather than exact logic and operates on **heuristic knowledge** and **heuristic rules**.

Example:

1. **One person can be at one place at a time.**
2. **If I put my hand in a fire, then it will burn.**

The above two statements are the examples of common sense reasoning which a human mind can easily understand and assume.

5. Monotonic Reasoning:

In monotonic reasoning, once the conclusion is taken, then it will remain the same even if we add some other information to existing information in our knowledge base. In monotonic reasoning, adding knowledge does not decrease the set of prepositions that can be derived. To solve monotonic problems, we can derive the valid conclusion from the available facts only, and it will not be affected by new facts. Monotonic reasoning is not useful for the real-time systems, as in real time, facts get changed, so we cannot use monotonic reasoning. Monotonic reasoning is used in conventional reasoning systems, and a logic-based system is monotonic. Any theorem proving is an example of monotonic reasoning.

Example:

- **Earth revolves around the Sun.**

It is a true fact, and it cannot be changed even if we add another sentence in knowledge base like, "The moon revolves around the earth" Or "Earth is not round," etc.

Advantages of Monotonic Reasoning:

- In monotonic reasoning, each old proof will always remain valid.
- If we deduce some facts from available facts, then it will remain valid for always.

Disadvantages of Monotonic Reasoning:

- We cannot represent the real world scenarios using Monotonic reasoning.
- Hypothesis knowledge cannot be expressed with monotonic reasoning, which means facts should be true.
- Since we can only derive conclusions from the old proofs, so new knowledge from the real world cannot be added.

6. Non-monotonic Reasoning

In Non-monotonic reasoning, some conclusions may be invalidated if we add some more information to our knowledge base. Logic will be said as non-monotonic if some conclusions can be invalidated by adding more knowledge into our knowledge base. Non-monotonic reasoning deals with incomplete and uncertain models. "Human perceptions for various things in daily life, "is a general example of non-monotonic reasoning.

Example: Let suppose the knowledge base contains the following knowledge:

- **Birds can fly**
- **Penguins cannot fly**
- **Pitty is a bird**

So from the above sentences, we can conclude that **Pitty can fly**. However, if we add one another sentence into knowledge base "**Pitty is a penguin**", which concludes "**Pitty cannot fly**", so it invalidates the above conclusion.

Advantages of Non-monotonic reasoning:

- For real-world systems such as Robot navigation, we can use non-monotonic reasoning.
- In Non-monotonic reasoning, we can choose probabilistic facts or can make assumptions.

Disadvantages of Non-monotonic Reasoning:

- In non-monotonic reasoning, the old facts may be invalidated by adding new sentences.
- It cannot be used for theorem **proving**.

Logical Reasoning

- **Logic** is a language of reasoning. It is a collection of rules called logic arguments, we use when doing logical reasoning.
- **Logical Reasoning** is a process of drawing conclusions from premises using rule of inference.

Formal logic.

- The formal logic is sometimes called **symbolic logic**.
- **Symbolic logic** is the study of **symbolic abstraction** (construct) that capture the formal features of logical inference by a formal system.

Informal Logic

- The informal logic is study of **natural language arguments**.
- The analysis of argument structures in ordinary language is part of informal logic
- The focus lies in distinguishing good arguments (valid) and bad arguments (invalid).

Example: 1

- Tanuj is older than Eina. Chetan is older than Tanuj. Eina is older than Chetan. If the first 2 statements are true, the 3rd statement is

  A. True
  B. False
  C. Uncertain

Sol: Option B

Explanation: If the first two statements are true, then Eina is the youngest of the three, so the third statement is necessarily false.

Example: 2

LOO, MON, NOM, OOL, ____

A. POK
B. HOL
C. HOK
D. JOI

Sol: Option A

Explanation: The 2nd letter is static, so concentrate on the 1st and 3rd letters. This series involves an alphabetical order with a reversal of the letters. The 1st letters are in alphabetical order: L, M, N, O & P. The 2nd and 4th segments are reversals of the 1st and 3rd segments. The missing segment begins with a new letter.

Example: 3

Pens cost more than pencils.
Pens cost less than eraser.
Erasers cost more than pencils and pens.
If the first two statements are true, the third statement is

A. True
B. False
C. Uncertain

Sol: Option A

Explanation: If the first two statements are true, then Erasers are the most expensive of the three stationery items.

Symbolic Reasoning under uncertainty

- ✓ Story so far
  - ✓ We have described techniques for reasoning with a complete, consistent and unchanging model of the world.
  - ✓ But in many problem domains, it is not possible to create such models. ***So here we are going to explore techniques for solving problems with incomplete and uncertain models.***

The ABC murder mystery example:

Let Abbott, Babbitt and Cabot be suspects in a murder case. Abbott has an alibi, in the register of a respected hotel in Albany. Babbitt also has an alibi, for his brother-in-law testified that Babbitt was visiting him in Brooklyn at the time. Cabot pleads alibi too, claiming to have been watching a ski meet in the cat skills.

We have only his words for that, So we believe

1. The Abbott did not commit the crime
2. The Babbitt did not commit the crime
3. The Abbott or Babbitt or Cabot did.

But presently Cabot documents his alibi-He had the good luck to have been caught by television in the sidelines at the ski meet. A new belief is thus trust upon us:

4. That Cabot did not.

- ✓ **Introduction to Non-monotonic Reasoning**
  - ✓ **Non monotonic reasoning:** in which the axioms and/or the rules of inference are extended to make it possible to reason with incomplete information.

These systems preserve, however, the property that , at any given moment, a statement is either believed to be true, believed to be false, or not believed to be either.

  - ✓ **Statistical Reasoning:**in which the representation is extended to allow some kind of **numeric measure of certainty**(rather than true or false) to be associated with each statement.
- ✓ At times we need to maintain many parallel belief spaces, each of which would correspond to the beliefs of one agent.
- ✓ Conventional reasoning systems, such as FOPL are designed to work with information that has three important properties.
- ✓ First-order logic is also known as **Predicate logic or First-order predicate logic**. First-order logic is a powerful language that develops information about the objects in a more easy way and can also express the relationship between those objects.
  - ✓ It is complete with respect to domain of interest.
  - ✓ It is consistent.
  - ✓ The only way it can change is that new facts can be added as they become available.
    - ✓ All this leads to monotonicity (these new facts are consistent with all the other facts that have already been asserted).
- ✓ At times we need to maintain many parallel belief spaces, each of which would correspond to the beliefs of one agent.
- ✓ Conventional reasoning systems, such as FOPL are designed to work with information that has three important properties.
- ✓ First-order logic is also known as **Predicate logic or First-order predicate logic**. First-order logic is a powerful language that develops information about the objects in a more easy way and can also express the relationship between those objects.
  - ✓ It is complete with respect to domain of interest.
  - ✓ It is consistent.
  - ✓ The only way it can change is that new facts can be added as they become available.

- ✓ All this leads to monotonicity (these new facts are consistent with all the other facts that have already been asserted).

- ✓ If any of these ***properties is not satisfied***, conventional logic based reasoning systems become inadequate. Non monotonic reasoning systems, are designed to be able to solve problems in which all of these properties may be missing
- ✓ Issues to be addressed
  - ✓ How can the knowledge base be ***extended*** to allow inferences to be made on the basis of lack of knowledge as well as on the presence of it?
  - ✓ How can the knowledge base be ***updated*** properly when a new fact is added to the system(or when the old one is removed)?
  - ✓ How can knowledge be used to help ***resolve conflicts*** when there are several in consistent non monotonic inferences that could be drawn?
- ✓ **Default Reasoning**
  - ✓ We use non monotonic reasoning to perform, what is commonly called Default Reasoning.
  - ✓ We want to draw conclusions based on what is most likely to be true.
  - ✓ Two approaches are
    - ✓ Non-monotonic Logic
    - ✓ Default Logic
  - ✓ Two common kinds of non-monotonic reasoning that can be defined in these logics :
    - ✓ Abduction
    - ✓ Inheritance

Non-monotonic Logic

- ✓ **Non Monotonic Logic(NML)**
  - ✓ It is one in which the language of FOPL is augmented with a modal operator M, which can be read as "is consistent."
  - ✓ Vx,y : Related(x,y) ^ M GetAlong(x,y) →WillDefend(x,y)
  - ✓ For all x and y, if x and y are related and if the fact that x gets along with y is consistent with everything else that is believed, then conclude that x will defend y.
  - ✓ If we are allowing statements in this form, one important issue that must be resolved if we want our theory to be even semi decidable, we must decide what "is consistent " means.

$$A \wedge \mathrm{M}\, B \rightarrow B$$
$$\neg A \wedge \mathrm{M}\, B \rightarrow B$$

we can derive the expression

$$\mathrm{M}\, B \rightarrow B$$

Default Logic

- ✓ **Default Logic**
  - ✓ An alternative logic for performing default-based reasoning is Reiter's Default Logic(DL) in which a new class of inference rules is introduced. In this approach, we allow inference rules of this form
    - ✓ A : B / C

The rule should be read as – If A is provable and it is consistent to assume B, then conclude C

Abduction

Abduction means systematic guessing: "infer" an assumption from a conclusion. For example, the following formula:

- ✓ Standard logic performs deductions. Given 2 axioms
  - ✓ Vx : A(x) → B(x)
  - ✓ A(C)
- ✓ We conclude B(C) using deduction

- ✓ Vx : Measels (x)→Spots(x)
- ✓ To conclude that if somebody has spots will surely have measels is incorrect, but it may be the best guess we can make about what is going on. Deriving conclusions in this way is this another form of default reasoning. We call this abductive reasoning.
- ✓ To accurately define abductive reasoning we may state that – Given 2 wff's A→B and B, for any expression A & B, if it is consistent to assume A, do so.

Rain example

"When it rains the grass gets weight the grasses with it must have rained"

means that it is using the conclusion and the rule to support that the precondition could explain the conclusion

Smoking house example

Fact : a large amount of black smoke is coming

Abduction 1: the house is on fire

Abduction 2 : to bad cook

Diagnosis example

Facts : 13 year old boy has a sharp pain in his right side, a fever, and a high white blood count

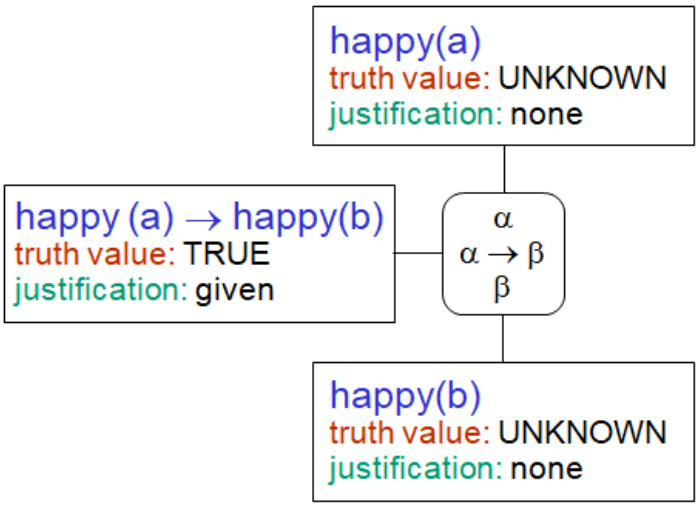
Abduction Inference: Appendicitis

Truth maintenance

demonstrated on one example

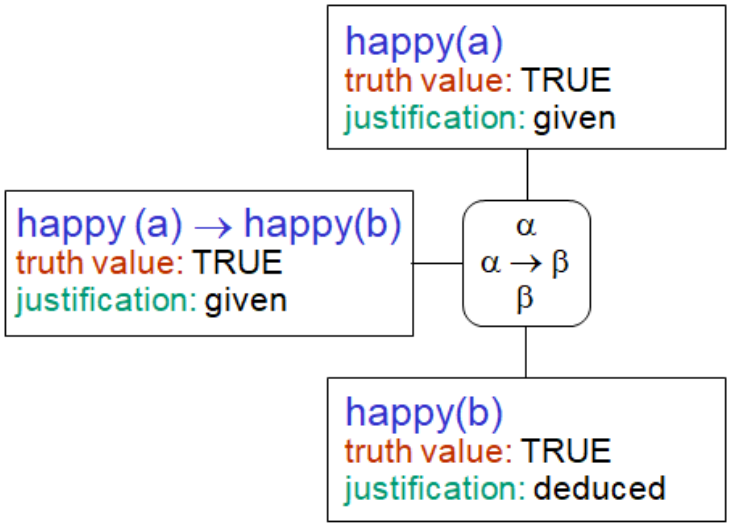
The first formula arrives:
we build a partial network.

happy(a) ⇒ happy(b)



A new fact: incorporate it
into the network.

happy(a)



Inheritance in Default Logic

Inheritance is a basis for inheriting attribute values from a prototype description of a class To individual entities that belong to the class. Inheritance says that "An object inherits attribute. Values from all the classes which it is member unless doing so leads to contradiction, in which case a value from a more restricted class has precedence over a value from a broader class".

Given :

$$\frac{Baseball\text{-}Player(x) : height(x, 6\text{-}1)}{height(x, 6\text{-}1)}$$

$$Pitcher(Three\text{-}Finger\text{-}Brown)$$

Conclude

$$height(Three\text{-}Finger\text{-}Brown, 6\text{-}1)$$

But this is blocked by

$$height(Three\text{-}Finger\text{-}Brown, 5\text{-}11)$$

Now we add:

$$\frac{Adult\text{-}Male(x) : height(x, 5\text{-}10)}{height(x, 5\text{-}10)}$$

Revised axiom :

$$\frac{Adult\text{-}Male(x) : \neg Baseball\text{-}Player(x) \wedge height(x, 5\text{-}10)}{height(x, 5\text{-}10)}$$

Minimalist Reasoning

- ✓ We describe methods for saying a very specific and highly useful class of things that are generally true. These methods are based on some variant of the idea of a minimal model.
- ✓ We will define a model to be minimal if there are no other models in which fewer things are true.
- ✓ The idea behind using minimal models as a basis for non-monotonic reasoning about the world is the following –
    - ✓ There are many fewer true statements than false ones. If something is true and relevant it makes sense to assume that it has been entered into

our knowledge base. Therefore, assume that the only true statements are those that necessarily must be true in order to maintain the consistency.

Two kinds of minimalist reasoning are:

(1) Closed world assumption (CWA).
(2) Circumscription.

The Closed World Assumption

The Closed World Assumption (CWA) states that the only true facts are those that are explicitly listed as true (in a knowledge base, in a database) or are provably true.

We may extend the theory by adding to it explicit negations of facts that cannot be proven.

In Prolog, this principle is implemented by "finite failure", or "negation as failure".

- ✓ This type of assumption is useful in application where most of the facts are true and it is, therefore, reasonable to assume that if a proposition cannot be proven, it is false.
- ✓ This is known as the closed world assumption with failure as negation. This means that in a kb if the ground literal p(a) is not provable, then ~p(a) is assumed to hold true.
    - ✓ CWA says that the only objects that satisfy any predicate P are those that must. Best for data base but some time fails with knowledge base.
    - ✓ Eg. A company's employee database.
    - ✓ Airline example

Essentially assume that everything you don't know (can't prove) is false.

Example: Travel agent database

Database of flights with origin and destination if a flight is not in the database assume it does not exist

Two advantages over CWA :

- ❖ Operates on whole formulas, not individual predicates.
- ❖ Allows some predicates to be marked as closed and others as open.

Accomplished by adding axioms that force a minimal interpretation on a selected portion of the KB.

Circumscription

Suppose that a child knows only two kinds of birds parrot and sparrow . Formally we can write this as

Bird (parrot) ∨ Bird (sparrow).

So, he defines a rule:

∀ X Bird (X) → Bird (parrot) ∨ Bird (sparrow).

∀ X Bird (X) → (X = parrot ∨ X = sparrow), → Predicate Expression

It uses the special predicate ab(x) which stands for Abnormal (x).

The interpretation of Ab(x) is intuitive (not formally defined) - an object is abnormal if it does not satisfy the rule

Uncertain Reasoning:

Statistical Methods

Symbolic versus statistical reasoning

The (Symbolic) methods basically represent uncertainty belief as being

- True,
- False, *or*
- Neither True nor False.

Some methods also had problems with

- Incomplete Knowledge
- Contradictions in the knowledge.

Statistical methods provide a method for representing beliefs that are not certain (or uncertain) but for which there may be some supporting (or contradictory) evidence. Statistical methods offer advantages in two broad scenarios:

Genuine Randomness

-- Card games are a good example. We may not be able to predict any outcomes with certainty but we have knowledge about the likelihood of certain items (*e.g.* like being dealt an ace) and we can exploit this.

Exceptions

-- Symbolic methods can represent this. However if the number of exceptions is large such system tend to break down. Many common sense and expert reasoning tasks for example. Statistical techniques can *summarise* large exceptions without resorting enumeration.

Basic Statistical methods -- Probability

The basic approach statistical methods adopt to deal with uncertainty is via the axioms of probability:

- Probabilities are (real) numbers in the range 0 to 1.

- A probability of $P(A) = 0$ indicates total uncertainty in A, $P(A) = 1$ total certainty and values in between some degree of (un)certainty.
- Probabilities can be calculated in a number of ways.

  Very Simply

  Probability = (number of desired outcomes) / (total number of outcomes)

  So given a pack of playing cards the probability of being dealt an ace from a full normal deck is 4 (the number of aces) / 52 (number of cards in deck) which is 1/13. Similarly the probability of being dealt a spade suit is 13 / 52 = 1/4.

  If you have a choice of number of items k from a set of items n then the formula $C_k^n = \frac{n!}{k!(n-k)!}$ is applied to find the number of ways of making this choice. (! = factorial).

  So the chance of winning the national lottery (choosing 6 from 49) is $\frac{49!}{6!43!} = 13,983,816$ to 1.

- Conditional probability, $P(A|B)$, indicates the probability of of event A given that we know event B has occurred.

Bayes Theorem

- This states:

$$P(H_i|E) = \frac{P(E|H_i)P(H_i)}{\sum_{k=1}^{n} P(E|H_k)P(H_k)}$$

  - This reads that given some evidence E then probability that hypothesis H_i is true is equal to the ratio of the probability that E will be true given H_i times the *a priori* evidence on the probability of H_i and the sum of the probability of E over the set of all hypotheses times the probability of these hypotheses.
  - The set of all hypotheses **must** be mutually exclusive and exhaustive.
  - Thus to find if we examine medical evidence to diagnose an illness. We must know all the prior probabilities of find symptom and also the probability of having an illness based on certain symptoms being observed.

Bayesian statistics lie at the heart of most statistical reasoning systems.

How is Bayes theorem exploited?

- The key is to formulate problem correctly:

  $P(A|B)$ states the probability of A given only B's evidence. *If* there is other relevant evidence then it **must** also be considered.

Herein lies a problem:

- All events must be *mutually exclusive*. However in real world problems events are not generally unrelated. For example in diagnosing measles, the symptoms of spots and a fever are related. This means that computing the conditional probabilities gets complex.

  In general if a prior evidence, *p* and some new observation, *N* then computing

$$P(H|N,p) = P(H|N)\frac{P(p|N,H)}{P(p|N)}$$

  grows exponentially for large sets of *p*

- All events must be *exhaustive*. This means that in order to compute all probabilities the set of possible events must be closed. Thus if new information arises the set must be created afresh and *all* probabilities recalculated.

Thus Simple Bayes rule-based systems are not suitable for uncertain reasoning.

- Knowledge acquisition is very hard.
- Too many probabilities needed -- too large a storage space.
- Computation time is too large.
- Updating new information is difficult and time consuming.
- Exceptions like ``none of the above" cannot be represented.
- Humans are not very good probability estimators.

Belief Models and Certainty Factors

This approach has been suggested by Shortliffe and Buchanan and used in their famous medical diagnosis MYCIN system.

MYCIN is essentially and *expert system*. Here we only concentrate on the probabilistic reasoning aspects of MYCIN.

- MYCIN represents knowledge as a set of rules.
- Associated with each rule is a *certainty factor*
- A certainty factor is based on measures of belief *B* and disbelief *D* of an hypothesis H_i given evidence *E* as follows:

$$B(H_i|E) = \begin{cases} 1 & \text{if } P(H_i) = 1 \\ \frac{\max[P(H_i|E),P(H_i)]-P(H_i)}{(1-P(H_i))P(H_i|E)} & \text{otherwise} \end{cases}$$

$$D(H_i|E) = \begin{cases} 1 & \text{if } P(H_i) = 0 \\ \frac{P(H_i)-\min[P(H_i|E),P(H_i)]}{P(H_i)P(H_i|E)} & \text{otherwise} \end{cases}$$

  where $P(H_i)$ is the standard probability.

- The certainty factor C of some hypothesis H_i given evidence E is defined as:
$$C(H_i|E) = B(H_i|E) - D(H_i|E).$$

Reasoning with Certainty factors

- Rules expressed as if *evidence list* $E_1, E_2, \ldots$ then there is suggestive evidence with probability, *p* for *symptom* H_i.
- MYCIN uses rules to reason backward to clinical data evidence from its goal of predicting a disease-causing organism.
- Certainty factors initially supplied by experts changed according to previous formulae.

- How do we perform reasoning when several rules are *chained* together?

  Measures of belief and disbelief given several observations are calculated as follows:

$$B(H_i|E_1, E_2) = \begin{cases} 0 & \text{if } D(H_i|E_1, E_2) = 1 \\ B(H_i|E_1) + B(H_i|E_2)(1 - B(H_i|E_1)) & \text{otherwise} \end{cases}$$

$$D(H_i|E) = \begin{cases} 0 & \text{if } B(H_i|E_1, E_2) = 1 \\ D(H_i|E_1) + D(H_i|E_2)(1 - D(H_i|E_1)) & \text{otherwise} \end{cases}$$

- How about our belief about several hypotheses taken together? Measures of belief given several hypotheses and to be combined logically are calculated as follows:

$$B(H_1 \wedge H_2|E) = \min(B(H_1|E), B(H_1|E)$$
$$B(H_1 \vee H_2|E) = \max(B(H_1|E), B(H_1|E)$$

  Disbelief is calculated similarly.

Fuzzy Logic

This topic is treated more formally in other courses. Here we summarise the main points for the sake completeness.

Fuzzy logic is a totally different approach to representing uncertainty:

- It focuses on ambiguities in describing events rather the uncertainty about the occurrence of an event.
- Changes the definitions of set theory and logic to allow this.
- Traditional set theory defines set memberships as a boolean predicate.

Fuzzy Set Theory

- *Fuzzy* set theory defines set membership as a *possibility distribution*.

  The general rule for this can expressed as:

$$f : [0,1]^n \rightarrow [0,1] .$$

where *n* some number of possibilities.

This basically states that we can take *n* possible events and us *f* to generate as single possible outcome.

This extends set membership since we could have varying definitions of, say, hot curries. One person might declare that only curries of Vindaloo strength or above are hot whilst another might say madras and above are hot. We could allow for these variations definition by allowing both possibilities in fuzzy definitions.

- Once set membership has been redefined we can develop *new logics* based on combining of sets *etc.* and reason effectively.

Knowledge Representation

When we collect knowledge we are faced with the problem of how to record it. And when we try to build the knowledge base we have the similar problem of how to represent it. We could just write down what we are told but, as the information grows, it becomes more and more difficult to keep track of therelationships between the items.

What to Represent

Facts: truths about the real world and what we represent. This can be regarded as the base knowledge level

Representation of the facts: which we manipulate. This can be regarded as the symbol level since we usually define the representation in terms of symbols that can be manipulated by programs.

- Simple way to store facts.

Simple Representation

Musician	Style	Instrument	Age
Miles Davis	Jazz	Trumpet	deceased
John Zorn	Avant Garde	Saxophone	35
Frank Zappa	Rock	Guitar	deceased
John Mclaughlin	Jazz	Guitar	47

- Each fact about a set of objects is set out systematically in columns.
- Little opportunity for inference.
- Knowledge basis for inference engines.

Weak Slot-and-Filler Structures

Why use this data structure?

- It enables attribute values to be retrieved quickly
- assertions are indexed by the entities
- binary predicates are indexed by first argument. *E.g. team(Mike-Hall , Cardiff).*

- Properties of relations are easy to describe .
- It allows ease of consideration as it embraces aspects of object oriented programming.
- So called because:
- A ***slot*** is an attribute value pair in its simplest form.
- A ***filler*** is a value that a slot can take -- could be a numeric, string (or any data type) value or a pointer to another slot.
- A *weak* slot and filler structure does not consider the *content* of the representation.

two types:

- Semantic Nets.
- Frames.

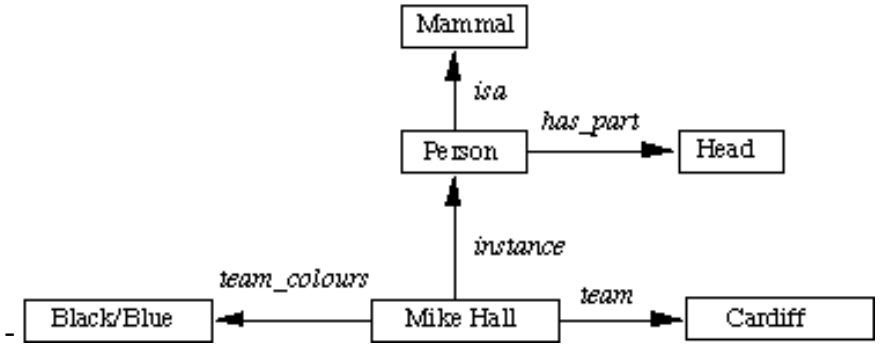
Semantic Nets

The major idea is that:

- The meaning of a concept comes from its relationship to other concepts, and that,
- The information is stored by interconnecting nodes with labelled arcs.

Representation in a Semantic Net

The physical attributes of a person can be represented

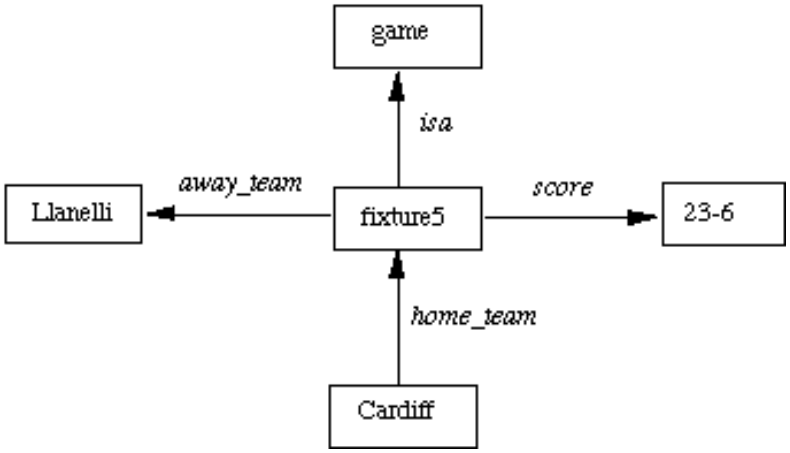


A Semantic Network

These values can also be represented in logic as: *isa(person, mammal), instance(Mike-Hall, person) team(Mike-Hall, Cardiff)*

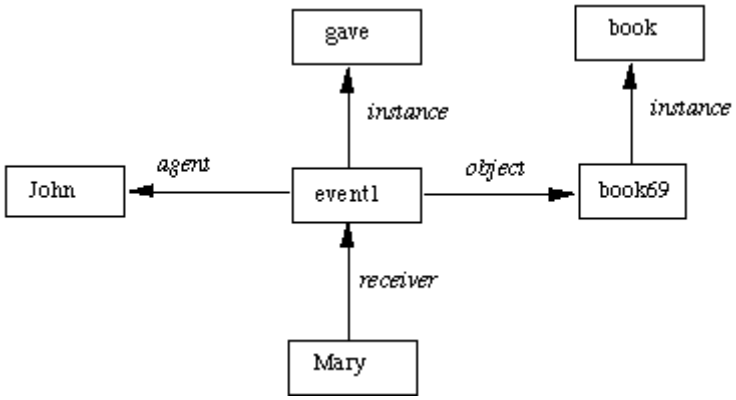
We have already seen how conventional predicates such as *lecturer(dave)* can be written as *instance (dave, lecturer)* Recall that *isa* and *instance* represent inheritance and are popular in many knowledge representation schemes. But we have a problem: *How we can have more than 2 place predicates in semantic nets? E.g. score(Cardiff, Llanelli, 23-6)* Solution:

- Create new nodes to represent new objects either contained or alluded to in the knowledge, *game* and *fixture* in the current example.
- Relate information to nodes and fill up slots



A Semantic Network for *n*-Place Predicate

As a more complex example consider the sentence: *John gave Mary the book*. Here we have several aspects of an event.



A Semantic Network for a Sentence

Inference in a Semantic Net

Basic inference mechanism: *follow links between nodes*.

Two methods to do this:

Intersection search

The notion that *spreading activation* out of two nodes and finding their intersection finds relationships among objects. This is achieved by assigning a special tag to each visited node.

Many advantages including entity-based organisation and fast parallel implementation. However very structured questions need highly structured networks.

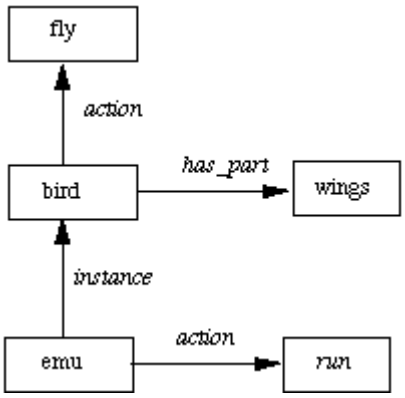
Inheritance

-- the *isa* and *instance* representation provide a mechanism to implement this.

Inheritance also provides a means of dealing with *default reasoning*. *E.g.* we could represent:

- Emus are birds.
- Typically birds fly and have wings.
- Emus run.

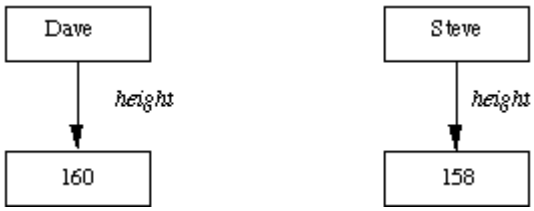
in the following Semantic net:



A Semantic Network for a Default Reasoning

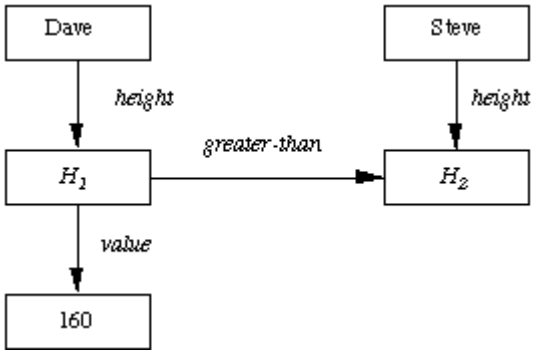
In making certain inferences we will also need to *distinguish between the link that defines a new entity and holds its value and the other kind of link that relates two existing entities*. Consider the example shown where the height of two people is depicted and we also wish to compare them.

We need extra nodes for the concept as well as its value.



Two heights

Special procedures are needed to process these nodes, but without this distinction the analysis would be very limited.



Comparison of two heights

Extending Semantic Nets

Here we will consider some extensions to Semantic nets that overcome a few problems (see Exercises) or extend their expression of knowledge.

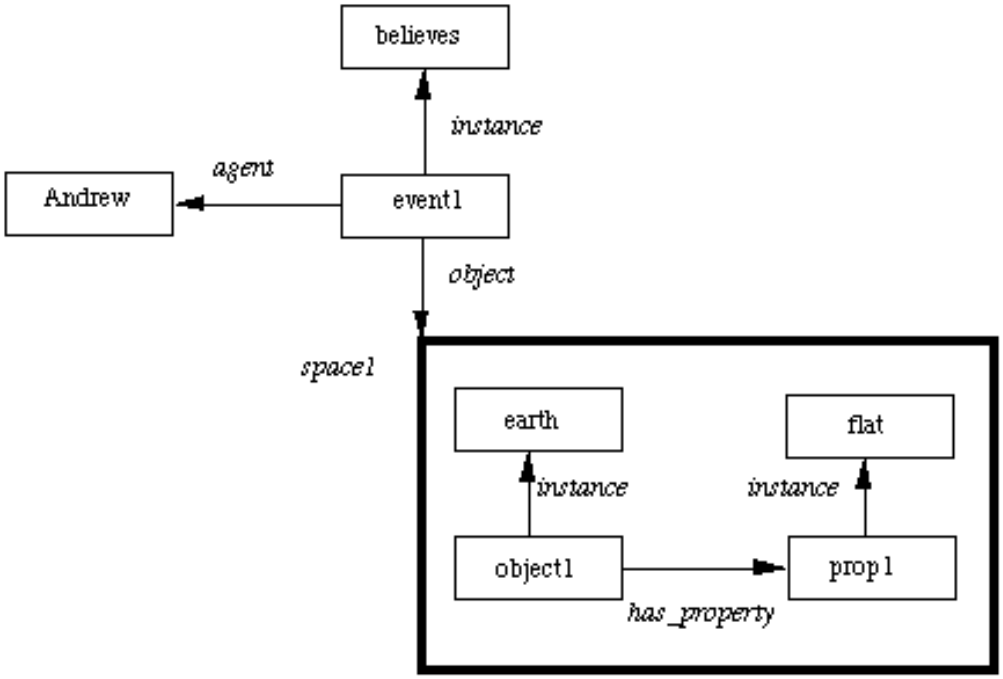
Partitioned Networks

Partitioned Semantic Networks allow for:

- propositions to be made without commitment to truth.
- expressions to be quantified.

Basic idea: *Break network into **spaces** which consist of groups of nodes and arcs and regard each **space** as a node.*

Consider the following: *Andrew believes that the earth is flat.* We can encode the proposition *the earth is flat* in a *space* and within it have nodes and arcs the represent the fact. We can have nodes and arcs to link this *space* the rest of the network to represent Andrew's belief.



Partitioned network

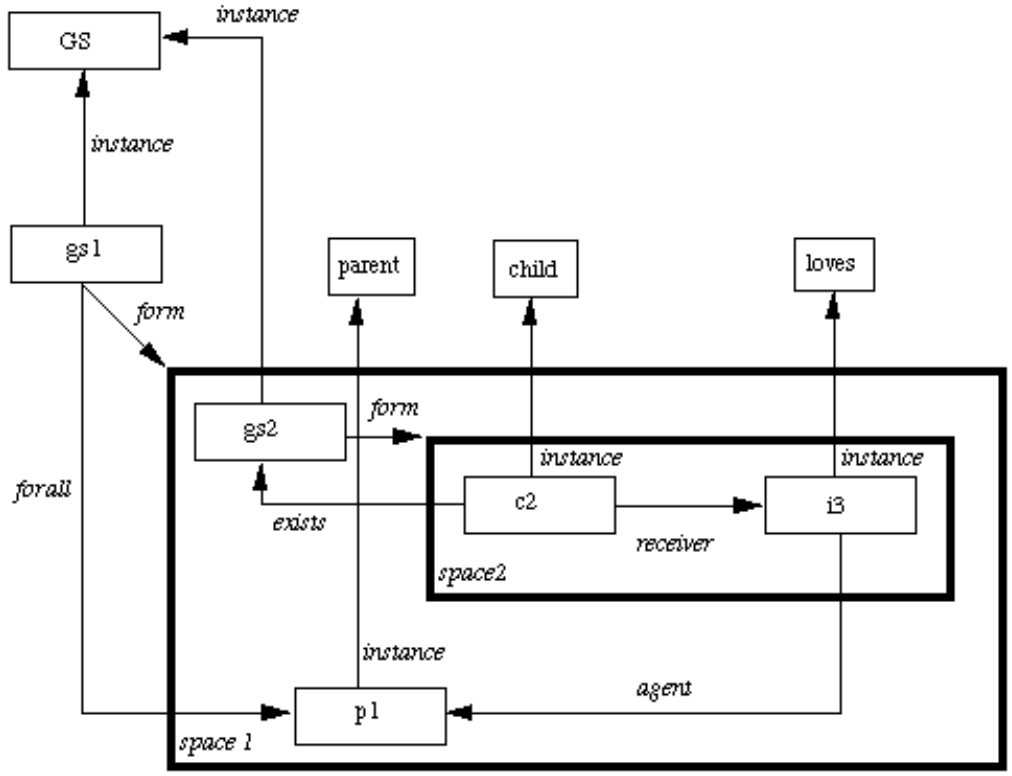
Now consider the quantified expression: *Every parent loves their child* To represent this we:

- Create a *general statement*, GS, special class.
- Make node g an instance of GS.
- Every element will have at least 2 attributes:
    - a *form* that states which relation is being asserted.
    - one or more *forall* (∀) or *exists* (∃) connections -- these represent universally quantifiable variables in such statements *e.g. x, y* in $\forall x : parent(x) \rightarrow \exists y : child(y) \wedge loves(x,y)$

Here we have to construct two *spaces one* for each *x,y*.

NOTE: We can express ∃ variables as *existentially qualified* variables and express the event of *love* having an agent *p* and receiver *b* for every parent *p* which could simplify the network (See Exercises).

Also If we change the sentence to *Every parent loves <u>child</u>* then the node of the object being acted on (*the child*) lies outside the form of the general statement. Thus it is not viewed as an existentially qualified variable whose value may depend on the agent. (See Exercises and Rich and Knight book for examples of this) So we could construct a partitioned network



∀Partitioned network

Advantage

- Semantic nets have the ability to represent default values for categories.
- Semantic nets convey some meaning in a transparent manner.
- Semantic nets are simple and easy to understand.
- Semantic nets are easy to translate into PROLOG.

Disadvantages

- Links between the objects represent only binary relations. For example, the sentence Run(ChennaiExpress, Chennai,Bangalore,Today) cannot be asserted directly.
- Some types of properties are not easily expressed using a semantic network. For example: negation, disjunction, and general non-taxonomic knowledge.
- There is no standard definition of link names.

Frames

- ✓ Frames can also be regarded as an extension to Semantic nets.
- ✓ Indeed it is not clear where the distinction between a semantic net and a frame ends. Semantic nets initially we used to represent labelled connections between objects.
- ✓ As tasks became more complex the representation needs to be more structured.
- ✓ The more structured the system it becomes more beneficial to use frames. A frame is a collection of attributes or slots and associated values that describe some real world entity.
- ✓ Frames on their own are not particularly helpful but frame systems are a powerful way of encoding information to support reasoning.

Set theory provides a good basis for understanding frame systems. Each frame represents:

- a class (set), or
- an instance (an element of a class).

Frame Knowledge Representation

Consider the example first discussed in Semantics Nets

Person

isa: *Mammal*

Cardinality: . . .

Adult-Male

isa: *Person*

Cardinality: . . .

Rugby-Player

isa: *Adult-Male*

Cardinality: . . .

Height:

Weight:

Position:

Team:

Team-Colours:

Back

isa:	*Rugby-Player*
Cardinality:	...
Tries:	

Mike-Hall

instance:	*Back*
Height:	*6-0*
Position:	*Centre*
Team:	*Cardiff-RFC*
Team-Colours:	*Black/Blue*

Rugby-Team

isa:	*Team*
Cardinality:	...
Team-size:	*15*
Coach:	

Cardiff-RFC

instance:	*Rugby-Team*
Team-size:	*15*
Coach:	*Terry Holmes*
Players:	{*Robert-Howley, Gwyn-Jones, ...* }

Figure: A simple frame system

Here the frames *Person, Adult-Male, Rugby-Player and Rugby-Team* are all **classes** and the frames *Robert-Howley* and *Cardiff-RFC* are instances.

Note

- The *isa* relation is in fact the subset relation.
- The *instance* relation is in fact *element of*.
- The *isa* attribute possesses a transitivity property. This implies: *Robert-Howley* is a *Back* and a *Back* is a *Rugby-Player* who in turn is an *Adult-Male* and also a *Person*.
- Both *isa* and *instance* have inverses which are called subclasses or all instances.
- There are attributes that are associated with the class or set such as cardinality and on the other hand there are attributes that are possessed by each member of the class or set.

DISTINCTION BETWEN SETS AND INSTANCES

It is important that this distinction is clearly understood. *Cardiff-RFC* can be thought of as a set of players or as an instance of a *Rugby-Team*.

If *Cardiff-RFC* were a *class* then

- its instances would be players
- it could not be a subclass of *Rugby-Team* otherwise its elements would be members of Rugby-Team which we do not want.

Instead we make it a subclass of *Rugby-Player* and this allows the players to inherit the correct properties enabling us to let the *Cardiff-RFC* to inherit information about teams.

This means that *Cardiff-RFC* is an instance of *Rugby-Team*.

BUT There is a problem here:

- A class is a set and its elements have properties.
- We wish to use inheritance to bestow values on its members.
- But there are properties that the set or class itself has such as the manager of a team.

This is why we need to view *Cardiff-RFC* as a subset of one class players and an instance of teams. We seem to have a CATCH 22. *Solution*: **Meta Classes**

A meta class is a special class whose elements are themselves classes. Now consider our rugby teams as:

Class	
instance:	*Class*
isa:	*Class*
Cardinality:	...
Team	
instance:	*Class*
isa:	*Class*
Cardinality:	{ The number of teams }
Team-Size:	*15*
Rugby-Team	
isa:	*Team*
Cardinality:	{ The number of teams }
Team-size: 15	
Coach:	
Cardiff-RFC	
instance:	*Rugby-Team*
Team-size:	*15*
Coach:	*Terry Holmes*
Robert-Howley	
instance:	*Back*
Height:	*6-0*
Position:	*Scrum Half*
Team:	*Cardiff-RFC*
Team-Colours:	*Black/Blue*

Figure: A Metaclass frame system

The basic metaclass is *Class*, and this allows us to

- define classes which are instances of other classes, and (thus)
- inherit properties from this class.

Inheritance of default values occurs when one element or class is an instance of a class.

Slots as Objects

How can we to represent the following properties in frames?

- Attributes such as *weight*, *age* be attached and make sense.
- Constraints on values such as *age* being less than a hundred
- Default values
- Rules for inheritance of values such as children inheriting parent's names
- Rules for computing values
- Many values for a slot.

A slot is a relation that maps from its domain of classes to its range of values. A relation is a set of ordered pairs so one relation is a subset of another. Since slot is a set the set of all slots can be represent by a meta class called *Slot*, say.

Consider the following:

SLOT

isa: *Class*

instance: *Class*

domain:

range:

range-constraint:

definition:

default:

to-compute:

single-valued:

Coach

instance: *SLOT*

domain: *Rugby-Team*

range: *Person*

range-constraint: λx *(experience x.manager)*

default:

single-valued: *TRUE*

Colour

instance: *SLOT*

domain: *Physical-Object*

range: *Colour-Set*

single-valued: *FALSE*

Team-Colours

instance: *SLOT*

isa: *Colour*

domain: *team-player*

range: *Colour-Set*

range-constraint: *not Pink*

single-valued: *FALSE*

Position

instance: *SLOT*

domain: *Rugby-Player*

range: *{ Back, Forward, Reserve }*

to-compute: λx*x.position*

single-valued: *TRUE*

NOTE the following:

- Instances of *SLOT* are slots
- Associated with *SLOT* are attributes that each instance will inherit.

- Each slot has a domain and range.
- Range is split into two parts one the class of the elements and the other is a constraint which is a logical expression if absent it is taken to be true.
- If there is a value for default then it must be passed on unless an instance has its own value.
- The *to-compute* attribute involves a procedure to compute its value. *E.g.* in *Position* where we use the dot notation to assign values to the slot of a frame.
- Transfers through lists other slots from which values can be derived from inheritance

Interpreting frames

A frame system interpreter must be capable of the following in order to exploit the frame slot representation:

- Consistency checking -- when a slot value is added to the frame relying on the domain attribute and that the value is legal using range and range constraints.
- Propagation of *definition* values along *isa* and *instance* links.
- Inheritance of default. values along *isa* and *instance* links.
- Computation of value of slot as needed.
- Checking that only correct number of values computed

Strong Slot and Filler Structures

Strong Slot and Filler Structures typically:

- Represent links between objects according to more *rigid* rules.
- Specific notions of what types of object and relations between them are provided.
- Represent knowledge about common situations.

Conceptual Dependency (CD)

Conceptual Dependency originally developed to represent knowledge acquired from natural language input.

The goals of this theory are:

- To help in the drawing of inference from sentences.
- To be independent of the words used in the original input.
- That is to say: *For any 2 (or more) sentences that are identical in meaning there should be only one representation of that meaning.*

It has been used by many programs that portend to understand English (*MARGIE, SAM, PAM*). CD developed by Schank *et al.* as were the previous examples.

CD provides:

- a structure into which nodes representing information can be placed
- a specific set of primitives
- at a given level of granularity.

Sentences are represented as a series of diagrams depicting actions using both abstract and real physical situations.

- The agent and the objects are represented
- The actions are built up from a set of primitive acts which can be modified by tense.

Examples of Primitive Acts are:

ATRANS

-- Transfer of an abstract relationship. *e.g. give*.

PTRANS

-- Transfer of the physical location of an object. *e.g. go.*

PROPEL

-- Application of a physical force to an object. *e.g. push*.

MTRANS

-- Transfer of mental information. *e.g. tell*.

MBUILD

-- Construct new information from old. *e.g. decide*.

SPEAK

-- Utter a sound. *e.g. say*.

ATTEND

-- Focus a sense on a stimulus. *e.g. listen, watch*.

MOVE

-- Movement of a body part by owner. *e.g. punch, kick*.

GRASP

-- Actor grasping an object. *e.g. clutch*.

INGEST

-- Actor ingesting an object. *e.g. eat*.

EXPEL

-- Actor getting rid of an object from body. *e.g. ????*.

Six primitive conceptual categories provide *building blocks* which are the set of allowable dependencies in the concepts in a sentence:

PP

-- Real world objects.

ACT

-- Real world actions.

PA

-- Attributes of objects.

AA

-- Attributes of actions.

T

-- Times.

LOC

-- Locations.

Arrows indicate the direction of dependency. Letters above indicate certain relationships:

o
-- object.

R

-- recipient-donor.

I

-- instrument *e.g. eat with a spoon*.

D

-- destination *e.g. going home.*

- Double arrows (⇔) indicate *two-way* links between the actor (PP) and action (ACT).
- The actions are built from the set of primitive acts (see above).
    - These can be modified by *tense etc.*

      The use of tense and mood in describing events is extremely important and schank introduced the following modifiers:

      p

-- past

f

-- future

t

-- transition

t_s

-- start transition

t_f

-- finished transition

k

-- continuing

?

-- interrogative

/

-- negative

delta

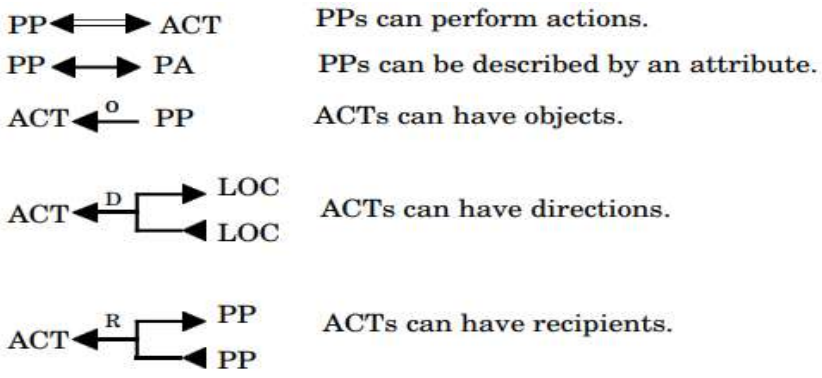
-- timeless

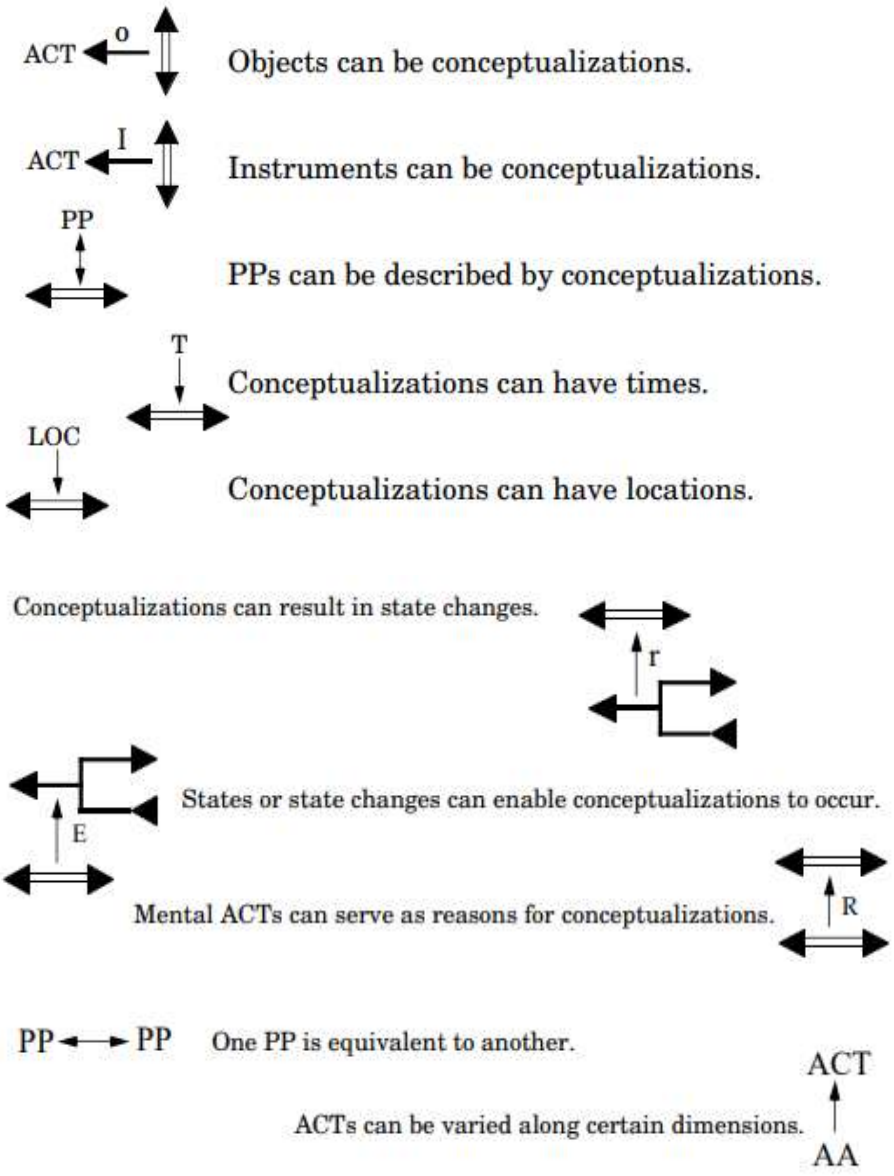
c

-- conditional

the absence of any modifier implies the *present tense*.

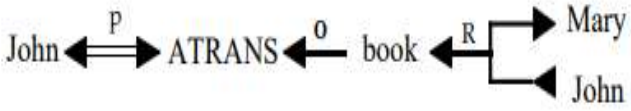
Conceptual Syntax Rules



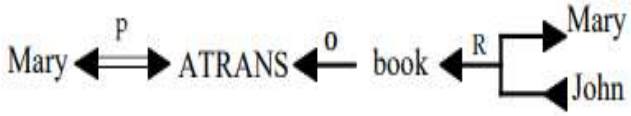


So the *past tense* of the above example:

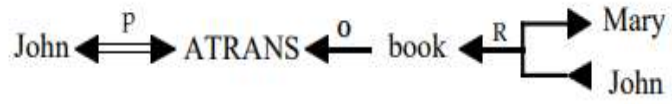
John gave Mary a book.



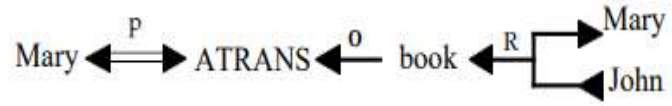
Mary took a book from John.



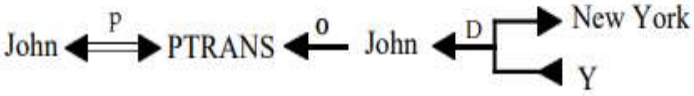
John gave Mary a book.



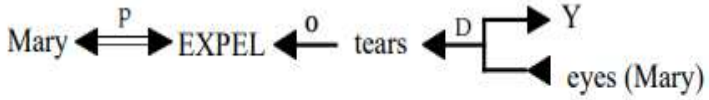
Mary took a book from John.



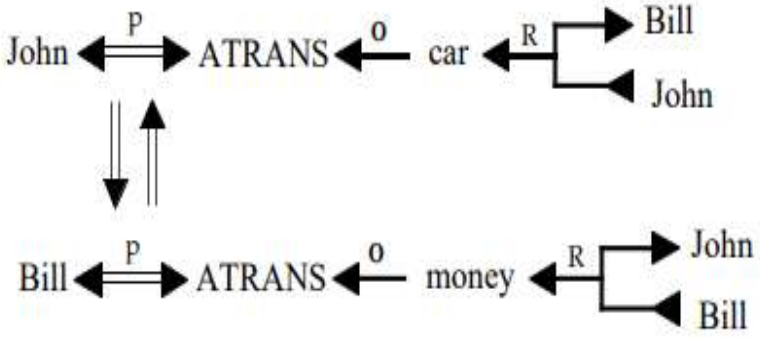
John went to New York.



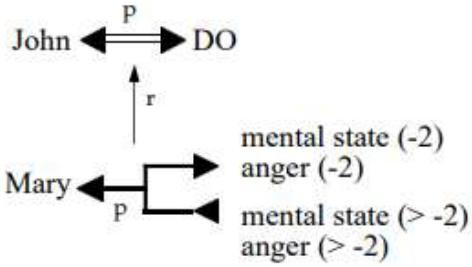
Mary cried.



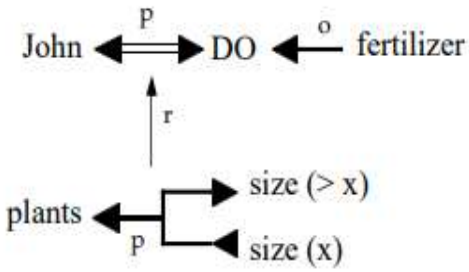
John sold his car to Bill



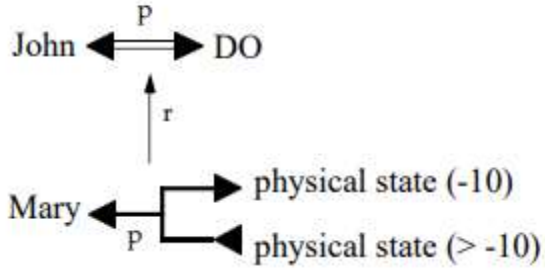
John annoyed Mary



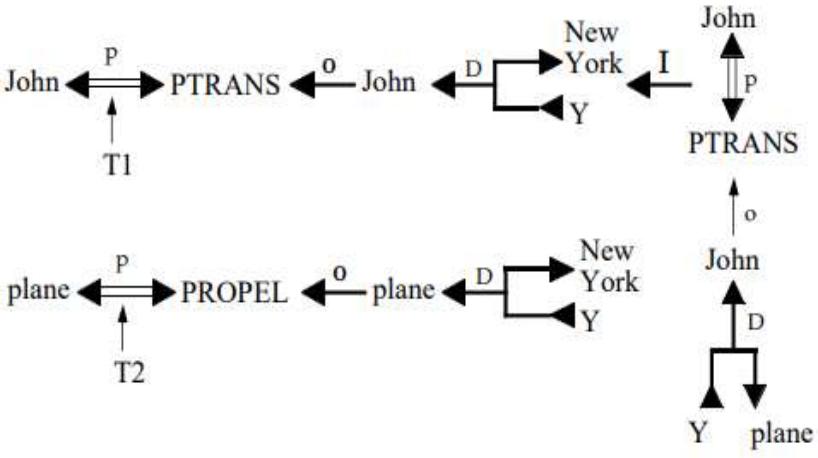
John grew the plants with fertilizer



John killed Mary



John flew to New York.



Advantages of CD:

- Using these primitives involves fewer inference rules.
- Many inference rules are already represented in CD structure.

Disadvantages of CD:

- Knowledge must be decomposed into fairly low level primitives.
- Impossible or difficult to find correct set of primitives.
- A lot of inference may still be required.
- Representations can be complex even for relatively simple actions.

Consider:

- *Dave bet Frank five pounds that Wales would win the Rugby World Cup*.
- Complex representations require a lot of storage

Applications of CD:

- MARGIE
  (*Meaning Analysis, Response Generation and Inference on English*) -- model natural language understanding.
- SAM
  (*Script Applier Mechanism*) -- Scripts to understand stories.
- PAM
  (*Plan Applier Mechanism*) -- Scripts to understand stories.

Scripts

- A *script* is a structure that prescribes a set of circumstances which could be expected to follow on from one another.
- It is similar to a thought sequence or a chain of situations which could be anticipated.
- It could be considered to consist of a number of slots or frames but with more specialised roles.

Scripts are beneficial because:

- Events tend to occur in known runs or patterns.
- Causal relationships between events exist.
- Entry conditions exist which allow an event to take place
- Prerequisites exist upon events taking place. *E.g.* when a student progresses through a degree scheme or when a purchaser buys a house.

The components of a script include:

- Entry Conditions
  -- these must be satisfied before events in the script can occur.
- Results
  -- Conditions that will be true after events in script occur.
- Props
  -- Slots representing objects involved in events.

- Roles
  -- Persons involved in the events.
- Track
  -- Variations on the script. Different tracks may share components of the same script.
- Scenes
  -- The sequence of *events* that occur. *Events* are represented in *conceptual dependency* form.

Scripts are useful in describing certain situations such as robbing a bank. This might involve:

- **Getting a gun.**
- **Hold up a bank.**
- **Escape with the money.**

Here the *Props* might be

- **Gun, *G*.**
- **Loot, *L*.**
- **Bag, *B***
- **Get away car, *C*.**

The *Roles* might be:

- **Robber, *S*.**
- **Cashier, *M*.**
- **Bank Manager, *O*.**
- **Policeman, *P*.**

The *Entry Conditions* might be:

- ***S* is poor.**
- ***S* is destitute.**

The *Results* might be:

- ***S* has more money.**
- ***O* is angry.**
- ***M* is in a state of shock.**
- ***P* is shot.**

There are 3 scenes: obtaining the gun, robbing the bank and the getaway.

Script: ROBBERY	*Track: Successful Snatch*
Props: G = Gun, L = Loot, B= Bag, C = Get away car.	*Roles*: R = Robber, M = Cashier, O = Bank Manager, P = Policeman.
Entry Conditions: R is poor. R is destitute.	*Results*: R has more money. O is angry. M is in a state of shock. P is shot.
Scene 1: Getting a gun R PTRANS R into Gun Shop R MBUILD R choice of G R MTRANS choice. R ATRANS buys G (go to scene 2)	
Scene 2 Holding up the bank R PTRANS R into bank R ATTEND eyes M, O and P R MOVE R to M position R GRASP G R MOVE G to point to M R MTRANS "Give me the money or ELSE" to M P MTRANS "Hold it Hands Up" to R R PROPEL shoots G P INGEST bullet from G M ATRANS L to M M ATRANS L puts in bag, B M PTRANS exit O ATRANS raises the alarm (go to scene 3)	
Scene 3: The getaway M PTRANS C	

Advantages of Scripts:

- Ability to predict events.
- A single coherent interpretation may be build up from a collection of observations.

Disadvantages:

- Less general than frames.
- May not be suitable to represent all kinds of knowledge.

CYC

- An ambitious attempt to form a very large knowledge base aimed at capturing commonsense reasoning.
- Initial goals to capture knowledge from a hundred randomly selected articles in the En*CYC*lopedia Britannica.
- Both Implicit and Explicit knowledge encoded.
- Emphasis on study of underlying information (assumed by the authors but not needed to tell to the readers.

We require special implicit knowledge or commonsense such as:

- We only die once.
- You stay dead.
- You cannot learn of anything when dead.
- Time cannot go backwards.

Why build large knowledge bases:

- **Brittleness**

-- Specialised knowledge bases are *brittle*. Hard to encode new situations and non-graceful degradation in performance. Commonsense based knowledge bases should have a firmer foundation.

- **Form and Content**

-- Knowledge representation may not be suitable for AI. Commonsense strategies could point out where difficulties in content may affect the form.

- **Shared Knowledge**

-- Should allow greater communication among systems with common bases and assumptions.

How is CYC coded?

- By hand.
- Special CYCL language:
  - LISP like.
  - Frame based
  - Multiple inheritance
  - Slots are fully fledged objects.
  - Generalised inheritance -- any link not just *isa* and *instance*.

UNIT III

KNOWLEDGE REPRESENTATION

What is knowledge representation?

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

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

What to Represent:

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

- **Object:** All the facts about objects in our world domain. E.g., Guitars contains strings, trumpets are brass instruments.
- **Events:** Events are the actions which occur in our world.
- **Performance:** It describe behavior which involves knowledge about how to do things.
- **Meta-knowledge:** It is knowledge about what we know.
- **Facts:** Facts are the truths about the real world and what we represent.
- **Knowledge-Base:** The central component of the knowledge-based agents is the knowledge base. It is represented as KB. The Knowledgebase is a group of the Sentences (Here, sentences are used as a technical term and not identical with the English language).

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

Types of knowledge

Following are the various types of knowledge:

1. Declarative Knowledge:

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

2. Procedural Knowledge

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

3. Meta-knowledge:

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

4. Heuristic knowledge:

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

5. Structural knowledge:

- Structural knowledge is basic knowledge to problem-solving.

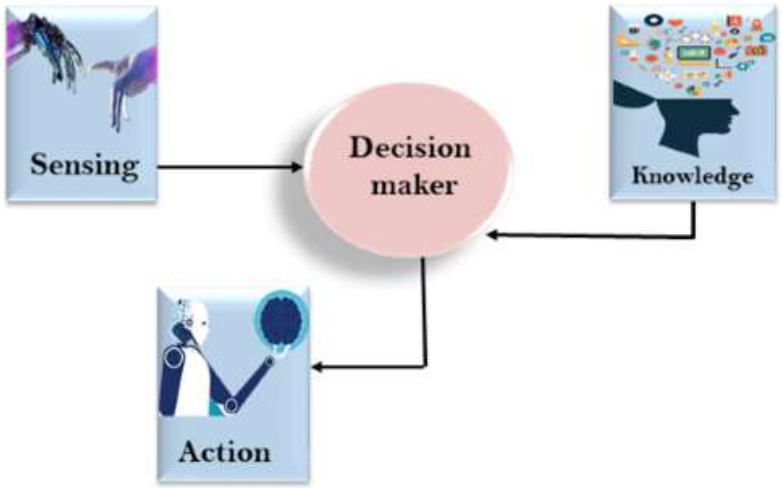
- It describes relationships between various concepts such as kind of, part of, and grouping of something.
- It describes the relationship that exists between concepts or objects.

The relation between knowledge and intelligence:

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

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

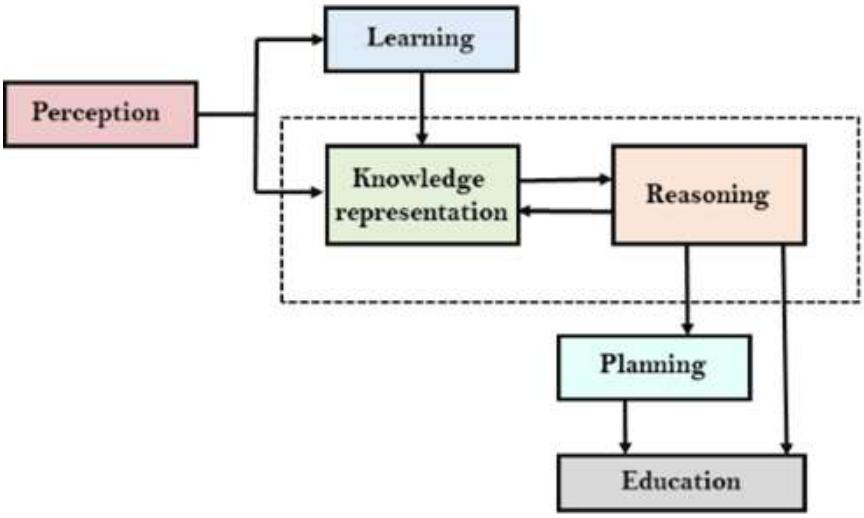
As we can see in below diagram, there is one decision maker which act by sensing the environment and using knowledge. But if the knowledge part will not present then, it cannot display intelligent behavior.



AI knowledge cycle:

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

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



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

Approaches to knowledge representation:

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

1. Simple relational knowledge:

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

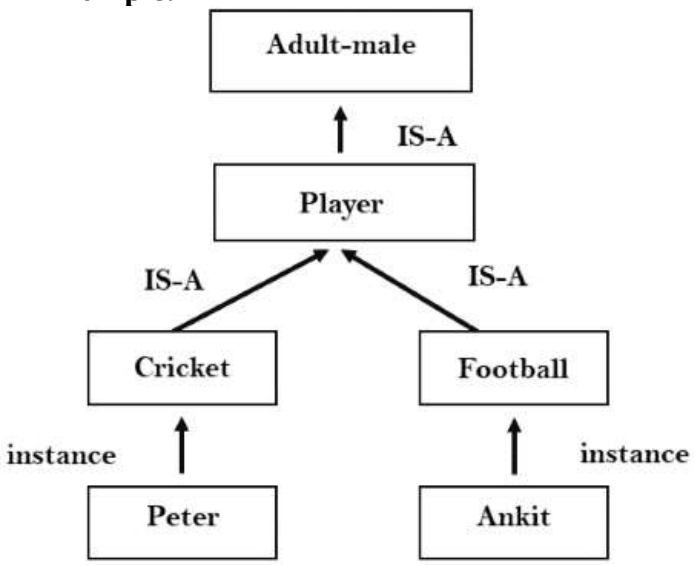
Example: The following is the simple relational knowledge representation.

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

2. Inheritable knowledge:

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

Example:



3. Inferential knowledge:

- Inferential knowledge approach represents knowledge in the form of formal logics.
- This approach can be used to derive more facts.
- It guaranteed correctness.
- **Example:** Let's suppose there are two statements:
    a. Marcus is a man
    b. All men are mortal
    Then it can represent as;

    man(Marcus)
    ∀x = man (x) ----------> mortal (x)s

4. Procedural knowledge:

- Procedural knowledge approach uses small programs and codes which describes how to do specific things, and how to proceed.
- In this approach, one important rule is used which is **If-Then rule**.
- In this knowledge, we can use various coding languages such as **LISP language** and **Prolog language**.
- We can easily represent heuristic or domain-specific knowledge using this approach.
- But it is not necessary that we can represent all cases in this approach.

Requirements for knowledge Representation system:

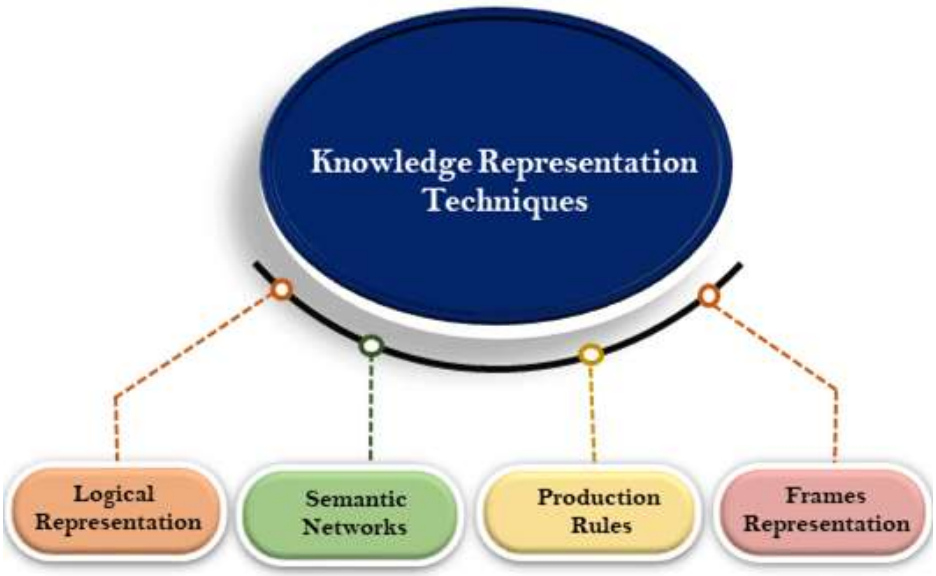
A good knowledge representation system must possess the following properties.

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

Techniques of knowledge representation

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

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



1. Logical Representation

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

Syntax:

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

- How to write those symbols.

Semantics:

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

Logical representation can be categorised into mainly two logics:

a. Propositional Logics
b. Predicate logics

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

Advantages of logical representation:

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

Disadvantages of logical Representation:

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

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

2. Semantic Network Representation

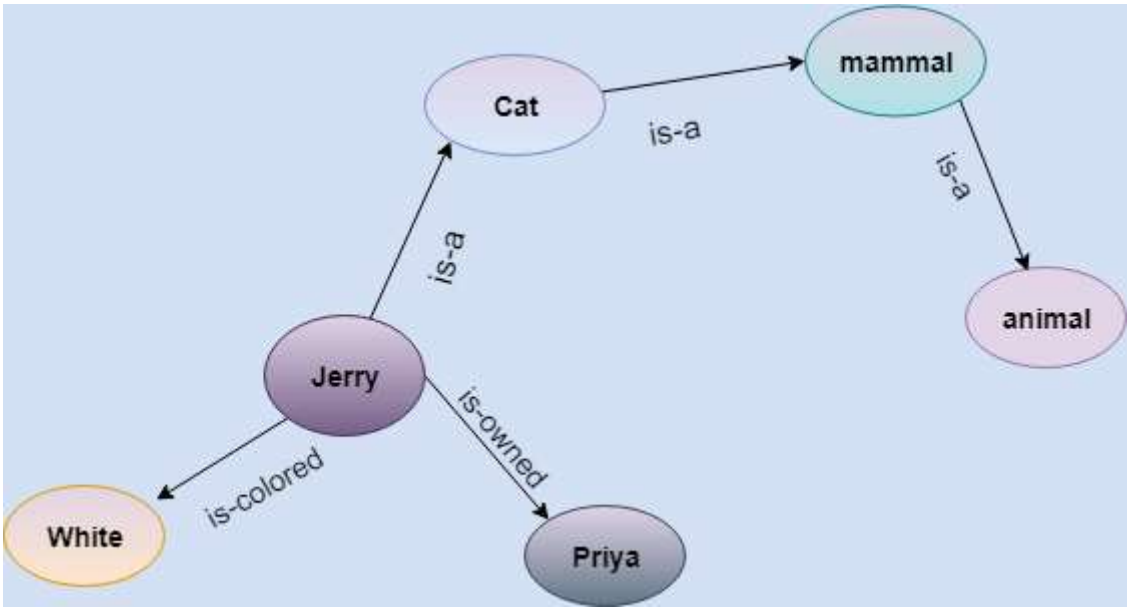
Semantic networks are alternative of predicate logic for knowledge representation. In Semantic networks, we can represent our knowledge in the form of graphical networks. This network consists of nodes representing objects and arcs which describe the relationship between those objects. Semantic networks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

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

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

Statements:

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



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

Drawbacks in Semantic representation:

1. Semantic networks take more computational time at runtime as we need to traverse the complete network tree to answer some questions. It might be possible in the worst case scenario that after traversing the entire tree, we find that the solution does not exist in this network.
2. Semantic networks try to model human-like memory (Which has 1015 neurons and links) to store the information, but in practice, it is not possible to build such a vast semantic network.
3. These types of representations are inadequate as they do not have any equivalent quantifier, e.g., for all, for some, none, etc.
4. Semantic networks do not have any standard definition for the link names.
5. These networks are not intelligent and depend on the creator of the system.

Advantages of Semantic network:

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

3. Frame Representation

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

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

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

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

Example: 1

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

Slots	Filters
Title	Artificial Intelligence
Genre	Computer Science
Author	Peter Norvig
Edition	Third Edition
Year	1996
Page	1152

Example 2:

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

Slots	Filter
Name	Peter
Profession	Doctor
Age	25
Marital status	Single
Weight	78

Advantages of frame representation:

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

Disadvantages of frame representation:

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

4. Production Rules

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

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

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

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

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

Example:

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

Advantages of Production rule:

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

Disadvantages of Production rule:

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

Knowledge representation issues

The fundamental goal of Knowledge Representation is to facilitate inferencing (conclusions) from knowledge. The issues that arise while using KR techniques are many. Some of these are explained below.

Issues that arises while KR techniques

1. **Important Attributes**
2. **Relationships among Attributes.**
3. **Choosing the granularity of Representation.**
4. **Representing Sets of Objects.**
5. **Finding the Right structures as Needed**

Important Attributes :

Any attribute of objects so basic that they occur in almost every problem domain ?

Relationship among attributes:

Any important relationship that exists among object attributes ?

Choosing Granularity :

At what level of detail should the knowledge be represented ?

Set of objects :

How sets of objects be represented ?

Finding Right structure :

Given a large amount of knowledge stored, how can relevant parts be accessed ?

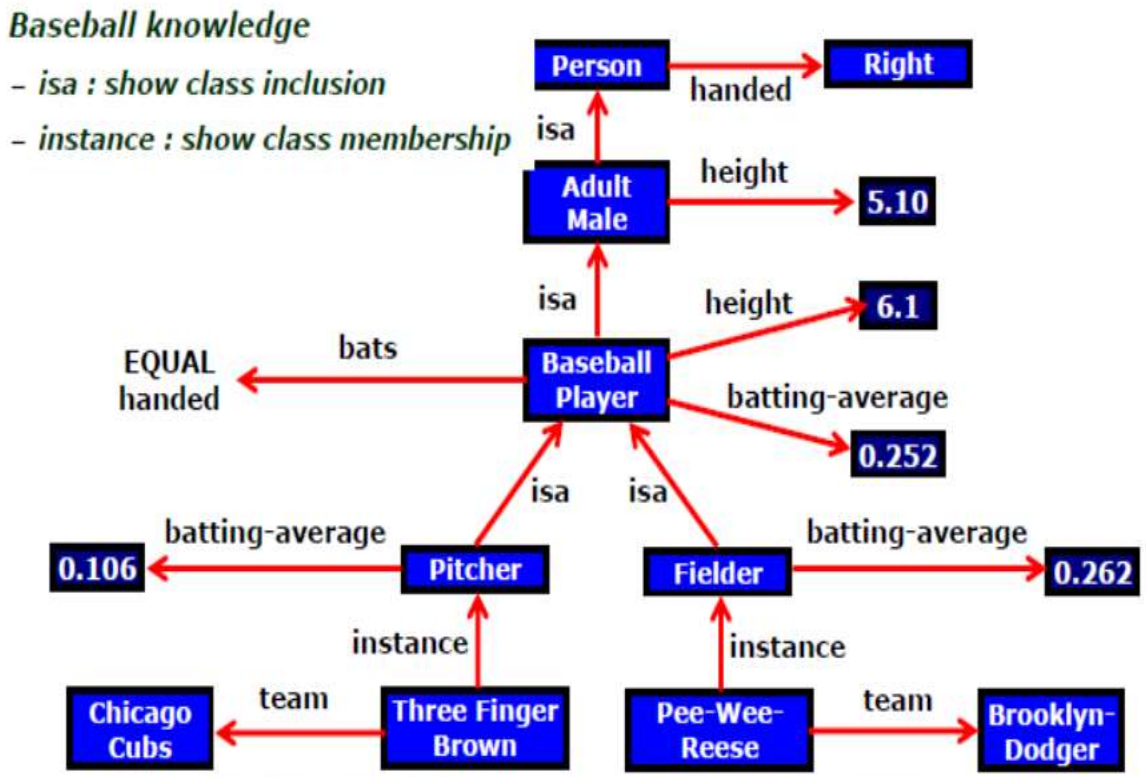


Fig. Inheritable knowledge representation (KR)

Important Attributes :

There are attributes that are of general significance. There are two attributes "**instance**" and "**isa**", that are of general importance. These attributes are important because they support *property inheritance*.

Relationship among Attributes

The attributes to describe objects are themselves entities they represent. The relationship between the attributes of an object, independent of specific knowledge they encode, may hold properties like:

> ***Inverses, existence in an isa hiera hy, techniques for reasoning about values and single valued attributes.***

Inverses :

This is about *consistency check*, while a value is added to one attribute. The entities are related to each other in many different ways. The figure shows attributes *(isa, instance, and team)*, each with a directed arrow, originating at the object being described and terminating either at the object or its value.

There are two ways of realizing this:

first, represent two relationships in a *single representation*; e.g., a logical representation, *team(Pee-Wee-Reese, Brooklyn–Dodgers)*, that can be interpreted as a statement about Pee-Wee-Reese or Brooklyn–Dodger.

second*, use attributes that focus on a *single entity *but use them in pairs*, one the inverse of the other; for e.g., one,

Dodgers , and the other, ***team = Pee-Wee-Reese, ...***
This second approach is followed in semantic net and frame-based systems, accompanied by a knowledge acquisition tool that guarantees the consistency of inverse slot by checking, each time a value is added to one attribute then the corresponding value is added to the inverse.

Existence in an "isa" hiera hy :

This is about *generalization-specialization*, like, classes of objects and specialized subsets of those classes. There are attributes and specialization of attributes.

Example: the attribute *"height*" is a specialization of general attribute

"physical-size" which is, in turn, a specialization of *"physical-attribute*". These generalization-specialization relationships for attributes are important because they support inheritance.

Techniques for reasoning about values :

This is about *reasoning values of attributes* not given explicitly. Several kinds of information are used in reasoning, like,

height : must be in a unit of length,

age : of person can not be greater than the age of person's parents.

The values are often specified when a knowledge base is created.

Single valued attributes :

This is about a *specific attribute* that is guaranteed to take a unique value.
Example : A baseball player can at time have only a single height and be a member of only one team. KR systems take different approaches to provide support for single valued attributes.

Choosing Granularity

What level should the knowledge be represented and what are the primitives ?

Should there be a small number or should there be a large number of low-level primitives or High-level facts.

High-level facts may not be adequate for inference while Low-level primitives may require a lot of storage.

Example of Granularity :
Suppose we are interested in following facts

John spotted Sue.
This could be represented as

Spotted (agent(John), object (Sue))
Such a representation would make it easy to answer questions such are

Who spotted Sue ?
Suppose we want to know

Did John see Sue ?
Given only one fact, we cannot discover that answer.

We can add other facts, such as

Spotted (x , y) → saw (x , y)
We can now infer the answer to the question.

Representing Sets of Objects
Certain properties of objects that are true as member of a set but not as individual;
Example : Consider the assertion made in the sentences
"there are more sheep than people in Australia", and "*English* speakers can be found all over the world."

To describe these facts, the only way is to attach assertion to the sets representing people, sheep, and English**.**

The reason to represent sets of objects is :

If a property is true for all or most elements of a set, then it is more efficient to associate it once with the set rather than to associate it explicitly with every elements of the set . This is done in different ways : in logical representation through the use of *universal quantifier*, and in hierarchal structure where node represent sets, the *inheritance propagate* set level assertion down to individual.

Example: assert **large (elephant)**;

Remember to make clear distinction between, whether we are asserting some property of the set itself, means, **the set of elephants is large**, or asserting some property that holds for individual elements of the set , means, **anything that is an elephant is large**.
There are three ways in which sets may be represented :

Name, as in the example – Ref Fig. Inheritable KR, the node - Baseball-Player and the predicates as Ball and Batter in logical representation.

Extensional definition is to list the numbers, and In tensional definition is to provide a rule, that returns true or false depending on whether the object is in the set or not.

Finding Right Structure

Access to right structure for describing a particular situation. It requires, selecting an initial structure and then revising the choice. While doing so, it is necessary to solve following problems: how to perform an initial selection of the most appropriate structure. How to fill in appropriate details from the current situations. How to find a better structure if the one chosen initially turns out not to be appropriate. what to do if none of the available structures is appropriate. When to create and remember a new structure. There is no good, general purpose method for solving all these problems. Some knowledge representation techniques solve some of them.

USING PREDICATE LOGIC

What is a Logic?

- **A language with concrete rules**
  - **No ambiguity in representation (may be other errors!)**
  - **Allows unambiguous communication and processing**
  - **Very unlike natural languages e.g. English**
- **Many ways to translate between languages**
  - **A statement can be represented in different logics**
  - **And perhaps differently in same logic**
- **Expressiveness of a logic**
  - **How much can we say in this language?**
- **Not to be confused with logical reasoning**
  - **Logics are languages, reasoning is a process (may use logic)**

Syntax and Semantics

- **Syntax**
  - **Rules for constructing legal sentences in the logic**
  - **Which symbols we can use (English: letters, punctuation)**
  - **How we are allowed to combine symbols**
- **Semantics**
  - **How we interpret (read) sentences in the logic**
  - **Assigns a meaning to each sentence**
- **Example: "All lecturers are seven foot tall"**
  - **A valid sentence (syntax)**

- **And we can understand the meaning (semantics)**
- **This sentence happens to be false (there is a counterexample)**

Propositional Logic

- **Syntax**
  - **Propositions, e.g. "it is wet"**
  - **Connectives: and, or, not, implies, iff (equivalent)**
  - $\wedge \quad \vee \quad \neg \quad \rightarrow \quad \leftrightarrow$
  - **Brackets, T (true) and F (false)**
- **Semantics (Classical AKA Boolean)**
  - **Define how connectives affect truth**
    - **"P and Q" is true if and only if P is true and Q is true**
  - **Use truth tables to work out the truth of statements**

Predicate Logic

- **Propositional logic combines atoms**
  - **An atom contains no propositional connectives**
  - **Have no structure (today_is_wet, john_likes_apples)**
- **Predicates allow us to talk about objects**
  - **Properties: is_wet(today)**
  - **Relations: likes(john, apples)**
  - **True or false**
- **In predicate logic each atom is a predicate**
  - **e.g. first order logic, higher-order logic**

First Order Logic

- **More expressive logic than propositional**
  - **Used in this course (representation in FOL)**
- **Constants are objects: john, apples**
- **Predicates are properties and relations:**
  - **likes(john, apples)**
- **Functions transform objects:**
  - **likes(john, fruit_of(apple_tree))**
- **Variables represent any object: likes(X, apples)**
- **Quantifiers qualify values of variables**
  - **True for all objects (Universal): ∀X. likes(X, apples)**
  - **Exists at least one object (Existential): ∃X. likes(X, apples)**

Example: FOL Sentence

- **"Every rose has a thorn"**
- $\forall X.\big(rose(X) \rightarrow \exists Y.(has(X,Y) \wedge thorn(Y))\big)$
- **For all X**
  - **if (X is a rose)**
  - **then there exists Y**
    - **(X has Y) and (Y is a thorn)**
- **"On Mondays and Wednesdays I go to John's house for dinner"**

$$\forall X.((is\_mon(X) \vee is\_wed(X)) \rightarrow eat\_meal(me, houseOf(john), X))$$

- **Note the change from "and" to "or"**
  - **Translating is problematic**

Higher Order Logic

- **More expressive than first order**
- **Functions and predicates are also objects**
  - **Described by predicates: binary(addition)**
  - **Transformed by functions: differentiate(square)**
  - **Can quantify over both**
- **E.g. define red functions as having zero at 17**
  $\forall F.(red(F) \leftrightarrow F(0) = 17)$
- **Much harder to reason with**

Logic is a Good Representation

- **Fairly easy to do the translation when possible**
- **Branches of mathematics devoted to it**
- **It enables us to do logical reasoning**
  - **Tools and techniques come for free**
- **Basis for programming languages**
  - **Prolog uses logic programs (a subset of FOL)**
  - **λProlog based on HOL**

Non-Logical Representations?

- **Production rules**
- **Semantic networks**
  - **Conceptual graphs**
  - **Frames**
- **Logic representations have restricitions and can be hard to work with**
  - **Many AI researchers searched for better representations**

Production Rules

- **Rule set of <condition,action> pairs**
  - **"if condition then action"**
- **Match-resolve-act cycle**
  - **Match: Agent checks if each rule's condition holds**
  - **Resolve:**
    - **Multiple production rules may fire at once (conflict set)**
    - **Agent must choose rule from set (conflict resolution)**
  - **Act: If so, rule "fires" and the action is carried out**
- **Working memory:**
  - **rule can write knowledge to working memory**

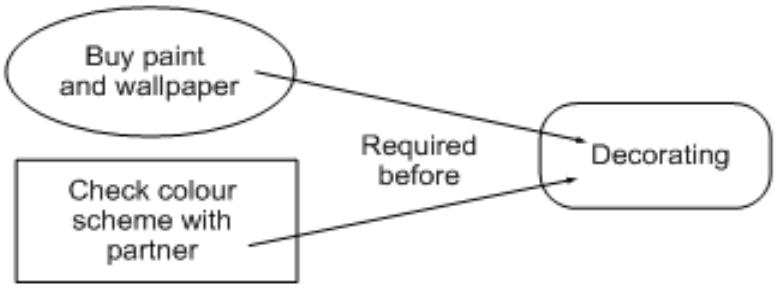
knowledge may match and fire other rules

Production Rules Example

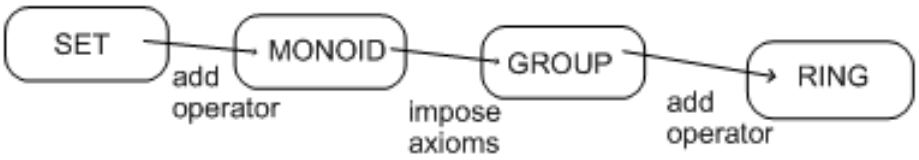
- **IF (at bus stop AND bus arrives) THEN action(get on the bus)**
- **IF (on bus AND not paid AND have oyster card) THEN action(pay with oyster) AND add(paid)**
- **IF (on bus AND paid AND empty seat) THEN sit down**
- **conditions and actions must be clearly defined**
  - **can easily be expressed in first order logic!**

Graphical Representation

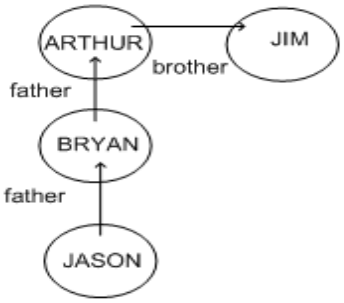
- **Humans draw diagrams all the time, e.g.**
  - **Causal relationships**

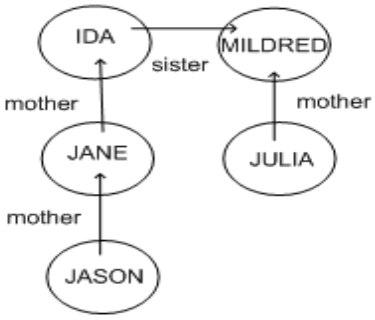


  - **And relationships between ideas**



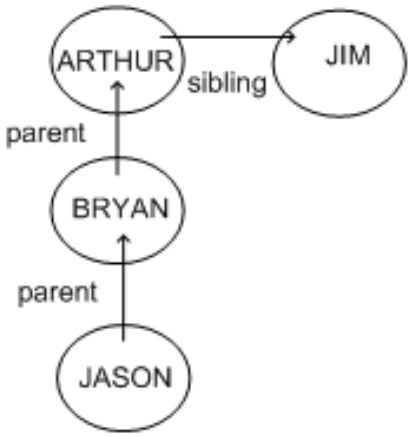
- **Graphs easy to store in a computer**
- **To be of any use must impose a formalism**

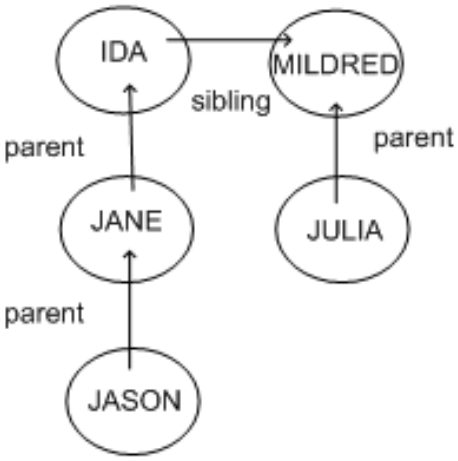




  - **Jason is 15, Bryan is 40, Arthur is 70, Jim is 74**
  - **How old is Julia?**

Semantic Networks

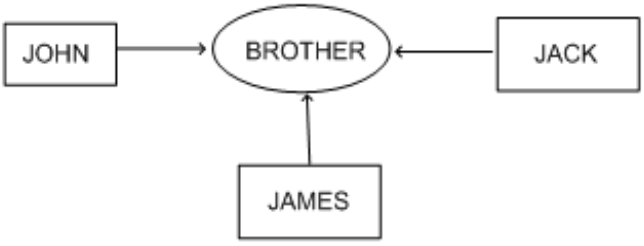




- **Because the syntax is the same**
  - **We can guess that Julia's age is similar to Bryan's**
- **Formalism imposes restricted syntax**
- **Graphical representation (a graph)**
  - **Links indicate subset, member, relation, ...**
- **Equivalent to logical statements (usually FOL)**
  - **Easier to understand than FOL?**
  - **Specialised SN reasoning algorithms can be faster**
- **Example: natural language understanding**
  - **Sentences with same meaning have same graphs**
  - **e.g. Conceptual Dependency Theory (Schank)**

Conceptual Graphs

- **Semantic network where each graph represents a single proposition**
- **Concept nodes can be**
  - **Concrete (visualisable) such as restaurant, my dog Spot**
  - **Abstract (not easily visualisable) such as anger**
- **Edges do not have labels**
  - **Instead, conceptual relation nodes**
  - **Easy to represent relations between multiple objects**

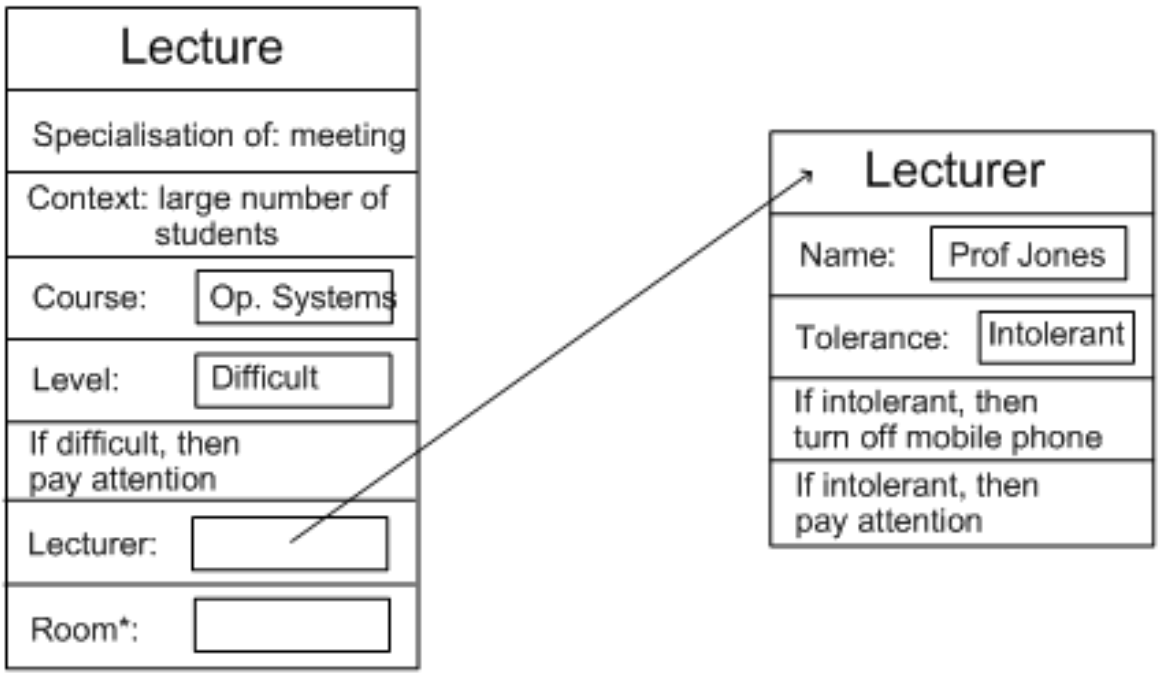


Frame Representations

- **Semantic networks where nodes have structure**
  - **Frame with a number of slots (age, height, ...)**
  - **Each slot stores specific item of information**
- **When agent faces a new situation**
  - **Slots can be filled in (value may be another frame)**
  - **Filling in may trigger actions**
  - **May trigger retrieval of other frames**
- **Inheritance of properties between frames**

- **Very similar to objects in OOP**

Example: Frame Representation



Representation & Logic

- **AI wanted "non-logical representations"**
  - **Production rules**
  - **Semantic networks**
    - **Conceptual graphs, frames**
- **But all can be expressed in first order logic!**
- **Best of both worlds**
  - **Logical reading ensures representation well-defined**
  - **Representations specialised for applications**
  - **Can make reasoning easier, more intuitive**

REPRESENTING KNOWLEDGE USING RULES

- **Rules are usually expressed in the form of IF . . . THEN . . . statements, such as:**
- **IF A THEN B This can be considered to have a similar logical meaning as the following: A→B**
- **A is called the antecedent and B is the consequent in this statement.**
- **In general, a rule can have more than one antecedent, usually combined either by AND or by OR (logically the same as the operators ∧ and ∨).**
- **Similarly, a rule may have more than one consequent, which usually suggests that there are multiple actions to be taken.**
- **In general, the antecedent of a rule compares an object with a possible value, using an operator.**
- **For example, suitable antecedents in a rule might be**
  - **IF x > 3**
  - **IF name is "Bob"**

    - **IF weather is cold**
- **Here, the objects being considered are x, name, and weather; the operators are ">" and "is", and the values are 3, "Bob," and cold.**
- **An example of a recommendation rule might be**
    - **IF name is "Bob"**
    - **AND weather is cold**
    - **THEN tell Bob 'Wear a coat'**
    - **It takes a set of inputs and gives advice as a result.**
- **The conclusion of the rule is actually an action, he should wear a coat.**

In some cases, the rules provide more definite actions such as "move left" or "close door," in which case the rules are being used to represent directives.

- **Rules can also be used to represent relations such as:**
    - **IF temperature is below 0**
    - **THEN weather is cold**

Rule-Based Systems

- **Rule-based systems or production systems are used to provide recommendations or diagnoses, or to determine a course of action in a particular situation or to solve a particular problem.**
- **A rule-based system consists of a number of components:**
    - **a database of rules (also called a knowledge base)**
    - **a database of facts**
    - **an interpreter, or inference engine**
- **knowledge base consists of a set of rules that represent the knowledge that the system has.**
- **database of facts represents inputs to the system that are used to derive conclusions, or to cause actions.**
- **interpreter, or inference engine, is the part of the system that controls the process of deriving conclusions. It uses the rules and facts, to draw conclusions.**
- **Using deduction to reach a conclusion from a set of antecedents is called forward chaining.**
- **An alternative method, backward chaining, starts from a conclusion and tries to show it by following a logical path backward from the conclusion to a set of antecedents that are in the database of facts.**

Forward Chaining

- **This is known as data-driven reasoning because the reasoning starts from a set of data and ends up at the goal.**
- **When applying forward chaining, the first step is to take the facts in the fact database and see if any combination of these matches all the antecedents of one of the rules in the rule database.**
- **When all the antecedents of a rule are matched by facts in the database, then this rule is triggered.**
- **Usually, when a rule is triggered, it is then fired, which means its conclusion is added to the facts database.**

For example, consider the following set of rules that is used to control an elevator in a three-story building:

- **Rule 1**
  IF on first floor and button is pressed on first floor
  THEN open door
- **Rule 2**
  IF on first floor
  AND button is pressed on second floor
  THEN go to second floor
- **Rule 3**
  IF on first floor
  AND button is pressed on third floor
  THEN go to third floor
- **Rule 4**
  IF on second floor
  AND button is pressed on first floor
  AND already going to third floor
  THEN remember to go to first floor later

- **This represents just a subset of the rules that would be needed, but we can use it to illustrate how forward chaining works.**
- **Let us imagine that we start with the following facts in our database:**
- **Fact 1**
  At first floor
- **Fact 2**
  Button pressed on third floor
- **Fact 3**
  Today is Tuesday
- **Now the system examines the rules and finds that Facts 1 and 2 match the antecedents of Rule 3. Hence, Rule 3 fires, and its conclusion "Go to third floor" is added to the database of facts.**
- **Note that Fact 3 was ignored altogether because it did not match the antecedents of any of the rules.**
- **Now let us imagine that the elevator is on its way to the third floor and has reached the second floor, when the button is pressed on the first floor. The fact Button pressed on first floor**

- **Is now added to the database, which results in Rule 4 firing.**

- **Now let us imagine that later in the day the facts database contains the following information:**
- **Fact 1**
  At first floor
- **Fact 2**
  Button pressed on second floor
- **Fact 3**
  Button pressed on third floor
- **In this case, two rules are triggered—Rules 2 and 3. In such cases where there is more than one possible conclusion, conflict resolution needs to be applied to decide which rule to fire.**

Conflict Resolution

- **In a situation where more than one conclusion can be deduced from a set of facts, there are a number of possible ways to decide which rule to fire.**
- **For example, consider the following set of rules:**
- **IF it is cold**
- **THEN wear a coat**
- **IF it is cold**
- **THEN stay at home**
- **IF it is cold**
- **THEN turn on the heat**
- **If there is a single fact in the fact database, which is "it is cold," then clearly there are three conclusions that can be derived. In some cases, it might be fine to follow all three conclusions, but in many cases the conclusions are incompatible.**
- **In one conflict resolution method, rules are given priority levels, and when a conflict occurs, the rule that has the highest priority is fired, as in the following example:**

- **IF patient has pain**
- **THEN prescribe painkillers priority 10**
- **IF patient has chest pain**
- **THEN treat for heart disease priority 100**
- **Here, it is clear that treating possible heart problems is more important than just curing the pain.**
- **An alternative method is the longest-matching strategy. This method involves firing the conclusion that was derived from the longest rule.**

For example:

- **IF patient has pain**
- **THEN prescribe painkiller**
- **IF patient has chest pain**
- **AND patient is over 60**
- **AND patient has history of heart conditions**
- **THEN take to emergency room**
- **Here, if all the antecedents of the second rule match, then this rule's conclusion should be fired.**
- **A further method for conflict resolution is to fire the rule that has matched the facts most recently added to the database.**

In each case, it may be that the system fires one rule and then stops, but in many cases, the system simply needs to choose a suitable ordering for the rules because each rule that matches the facts needs to be fired at some point.

Meta Rules

- **In this case, it clearly makes more sense to allow the earlier rules priority.**
- **This kind of knowledge is called meta knowledge—knowledge about knowledge. The rules that define how conflict resolution will be used, and how other aspects of the system itself will run, are called meta rules.**
- **The knowledge engineer who builds the expert system is responsible for building appropriate meta knowledge into the system (such as "expert A is to be trusted more than expert B" or "any rule that involves drug X is not to be trusted as much as rules that do not involve X").**

Meta rules are treated by the expert system as if they were ordinary rules but are given greater priority than the normal rules that make up the expert system.

Backward Chaining

- **Forward chaining can be inefficient because it may end up proving a number of conclusions that are not currently interesting.**
- **In such cases, where a single specific conclusion is to be proved, backward chaining is more appropriate.**
- **In backward chaining, we start from a conclusion, which is the hypothesis we wish to prove, and we aim to show how that conclusion can be reached from the rules and facts in the database.**

- **The conclusion we are aiming to prove is called a goal, and so reasoning in this way is known as goal-driven reasoning.**
- **Backward chaining is often used in formulating plans.**

- **A plan is a sequence of actions that a program decides to take to solve a particular problem.**
- **Backward chaining in this way starts with the goal state, which is the set of conditions the agent wishes to achieve in carrying out its plan. It now examines this state and sees what actions could lead to it.**
- **For example, if the goal state involves a block being on a table, then one possible action would be to place that block on the table.**
- **Backward chaining ensures that each action that is taken is one that will definitely lead to the goal, and in many cases this will make the planning process far more efficient.**

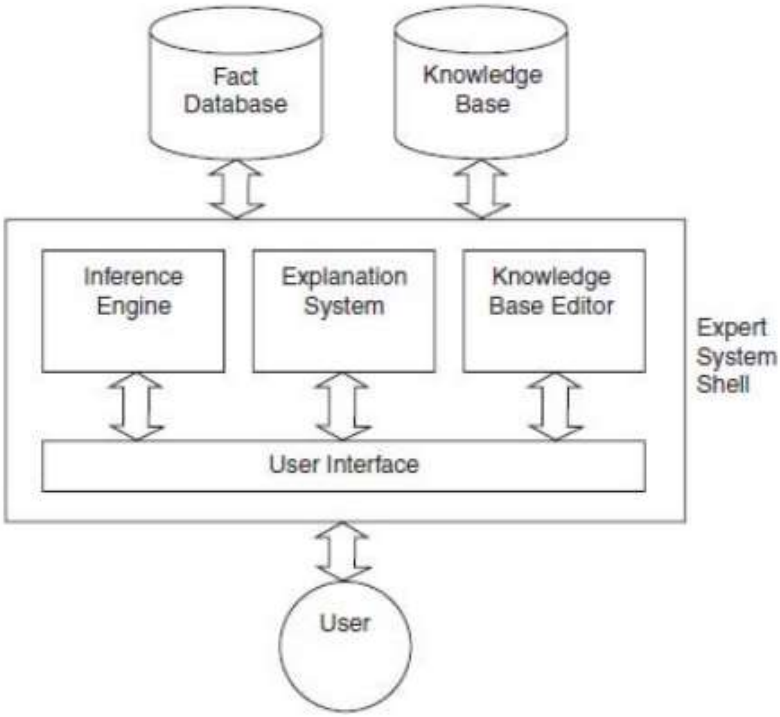
Rule-Based Expert Systems

- **An expert system is one designed to model the behaviour of an expert in some field, such as medicine or geology.**
- **Rule-based expert systems are designed to be able to use the same rules that the expert would use to draw conclusions from a set of facts that are presented to the system.**

The People Involved in an Expert System

- **The end-user of the system is the person who has the need for the system.**
- **In the case of a medical diagnosis system, this may be a doctor, or it may be an individual who has a complaint that they wish to diagnose.**
- **The knowledge engineer is the person who designs the rules for the system, based on either observing the expert at work or by asking the expert questions about how he or she works.**
- **The domain expert is very important to the design of an expert system. In the case of a medical diagnosis system, the expert needs to be able to explain to the knowledge engineer how he or she goes about diagnosing illnesses.**

Architecture of an Expert System



- **The knowledge base contains the specific domain knowledge that is used by an expert to derive conclusions from facts.**
- **In the case of a rule-based expert system, this domain knowledge is expressed in the form of a series of rules.**
- **The explanation system provides information to the user about how the inference engine arrived at its conclusions.**
- **The fact database contains the case-specific data that are to be used in a particular case to derive a conclusion.**
- **The inference engine is the part of the system that uses the rules and facts to derive conclusions. The inference engine will use forward chaining, backward chaining, or a combination of the two to make inferences from the data that are available to it.**
- **The knowledge-base editor allows the user to edit the information that is contained in the knowledge base.**
- **The knowledge-base editor is not usually made available to the end user of the system but is used by the knowledge engineer or the expert to provide and update the knowledge that is contained within the system.**

The Expert System Shell

- **the expert system that do not contain domain-specific or case-specific information are contained within the expert system shell.**
- **This shell is a general toolkit that can be used to build a number of different expert systems, depending on which knowledge base is added to the shell.**

Syntactic- Semantic of Representation

Knowledge Representation

- **Basis of each AI concept or system!**

- **Representation without processing makes no sense (therefore we started with knowledge processing)**
- **Same knowledge can be represented very differently:**
  - **Spectrum: computer friendly - human friendly**
  - **Levels of abstraction**
  - **Different views on problem**
  - **Different processing techniques**

Note: transformations are possible!

- **What is representation?**

Representation refers to a symbol or thing which represents (’refers to’, ’stands for’) something else.

- **When do we need to represent?**

We need to represent a thing in the natural world when we don’t have, for some reason, the possibility to use the original ’thing’.

- **The object of knowledge representation is to express the problem in computer-understandable form**

Aspects of KR

Syntactic

- **Possible (allowed) constructions**
- **Each individual representation is often called a sentence.**
- **For example: color(my_car, red), my_car(red), red(my_car), etc.**

Semantic

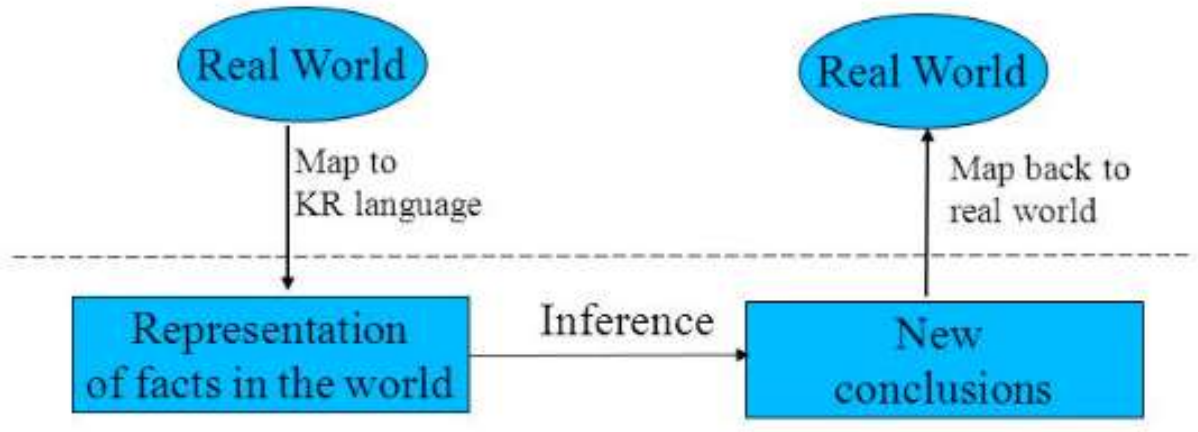
- **What does the representation mean (maps the sentences to the world)**
- **For example: color(my_car, red) → ??**
  ‘my car is red’, ‘paint my car red’, etc.

Inferential

- **The interpreter**
- **Decides what kind of conclusions can be drawn**
- **For example: Modus ponens (P, P→Q, therefore Q)**

Well-defined syntax/semantics

- **Knowledge representation languages should have precise syntax and semantics.**
- **You must know exactly what an expression means in terms of objects in the real world.**



Declarative vs. procedural knowledge

- **Declarative knowledge (facts about the world)**
  - **A set of declarations or statements.**
  - **All facts stated in a knowledge base fall into this category of knowledge.**
  - **In a sense, declarative knowledge tells us what a problem (or problem domain) is all about**
  - **Example:**

cheaper(coca_cola, pepsi)
tastier(coca_cola, pepsi)
if (cheaper(x,y) && (tastier(x,y)) → buy(x)

- **Procedural knowledge (how something is done)**
  - **Something that is not stated but which provides a mean of dynamically (usually at run-time) arriving at new facts.**
  - **Example: shopping script**

Syntax and Semantics

- **Similar to programming languages, in knowledge representation we have to look at syntax and semantics of a representation approach**
- **Syntax: What symbols, data types, etc. are allowed; sorts, number of arguments (multiplicity) and so on? What symbols have special meaning (and therefore have to be used with this meaning in mind)?**
- **Semantics: What do the symbols mean, what has knowledge processing to accomplish?**

Logics

- **Considered by humans as the knowledge representation (and processing) method of computers**
- **Clear mathematical foundation:**

syntax describes formulas; axioms what is considered true; inference rules how to get other true formulas

- **Many different kinds of logics**
- **Meaning of a formula usually not easy to determine by humans (rather formal semantics)**

Propositional logic

General idea:

- **Formulas describe combinations of statements (propositions) that are either truth or false and this way build statements themselves.**
- **No parameterized statements!**
- **Basis of the logics of gates, circuits and micro chips**

Semantics

- Look for tautologies, i.e. formulas that are interpreted to true by all $I \in \mathcal{I}$
- $I(\neg p)$ = true, if $I(p)$ = false; false else
- $I(p \vee q)$ = true, if $I(p)$ or $I(q)$ = true; false else
- $I(p \wedge q)$ = true, if $I(p)$ and $I(q)$ = true; false else
- $I(p \rightarrow q)$ = false, if $I(p)$ = true and $I(q)$ = false; true else
- $I(p \leftrightarrow q)$ = true, if $I(p) = I(q)$; false else

How to get knowledge into the representation structure

- assign predicate symbols to simple positive statements
- Connect them to form complicated statements
- But be careful: "tertium non datur"
  - The car is green =: p
  - The car is red =: q
  - We need in addition: $q \leftrightarrow \neg p$

Discussion

+ decidable, but NP complete
- not very expressive
- knowledge bases get very large

And what about processing data?

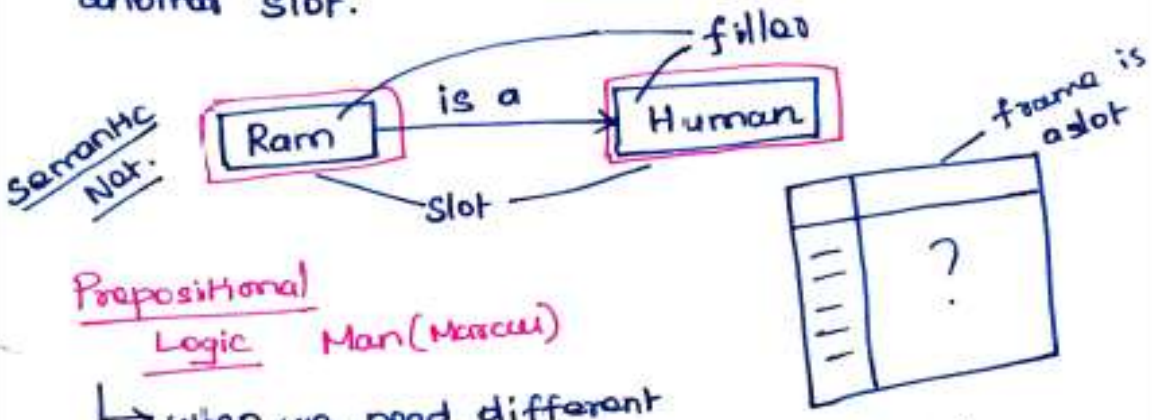
- Calculus used in most (best) systems:
  Davis-Putnam (working on clauses; special case of Modelelimination)
- Each formula can be transformed into equivalent set of clauses (remember: formula with J = {¬, ∨})
  - "defining" equations for → and ↔
  - DeMorgan's laws to move negation inward
- For deciding tautologies, we use and-tree-based search
- For testing for satisfiability, we see clauses as constraints and use or-tree-based search

Example

- Represent the following statements in propositional logic:
  - A Ferrari is a red car.
  - Red cars are fast cars.
  - Bad cars are slow cars.
- Show that the following statement is a logical consequence of the statements above:
  - A Ferrari is a good car.

WEAK SLOT AND FILLER STRUCTURES

* A Slot is an attribute value pair in its simplest form.
* A filler is a value that a slot can take, it can be numeric, string or pointer to another slot.

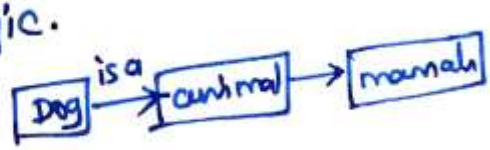


Propositional Logic Man(Marcus)

↳ when we need different form of Logic then we need this structure.

When we need this?

→ Monotonic inheritance can be perform substantially more efficiently with these structure than with pure logic.



Dog is an animal

Animals are mamals.

↓

mamals(animals(Dog)) → Not proper representation.

– So we need a new structure to represent, Monotonic inheritance, ~~make this~~ easy.

→ The reason that makes inheritance easy is the knowledge in slot and filler structure system is structured as a set of entities and their attributes.

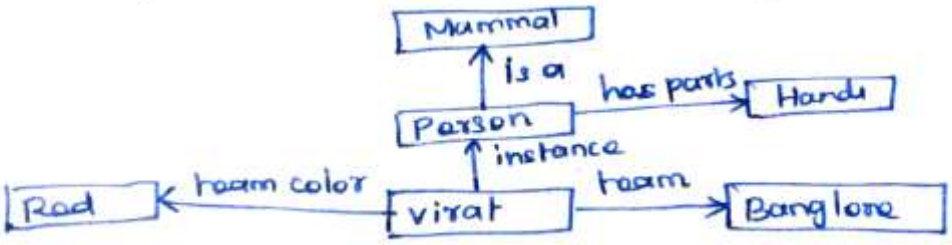
2 types:

1) Semantic Net
2) Frames

Semantic Net:

* an alternative to predicate logic as a form of knowledge representation.

* The idea is that we can store our knowledge in the form of graph, with node representing objects in the world and the arcs representing relationships b/w these object.



* Inheritance → (Predicate logic tough) association (is a relation) can be described using semantic N/w.

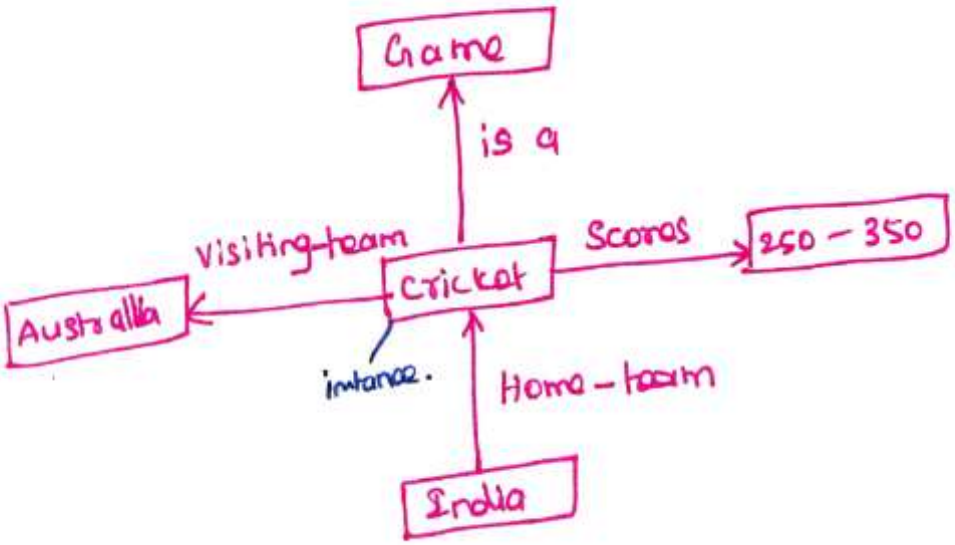
* Semantic N/w provide direct indexing for objects, categories and the link b/w them.

* Binary relationship: → predicate logic
is a (mammal, people)

→ Simple Binary predicate can be represented but other can be represented using is a & instance.

→ three or more predicate can also be converted by creating one new object representing entire predicate stmt.

eg: Avgscore (Australia, India, 250-350)

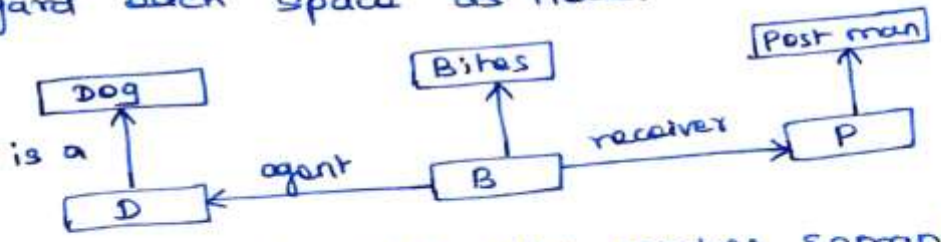


Partitioned N/W:

→ Partitioned semantic N/W's allows to represent difference b/w description of an individual object or process and description of set of objects.

→ The set description involve quantification.

→ Basic Idea: Break N/W into spaces which consist of groups of nodes and arcs and regard each space as node.



→ Partition of semantic N/W makes semantic N/W logically more adequate.

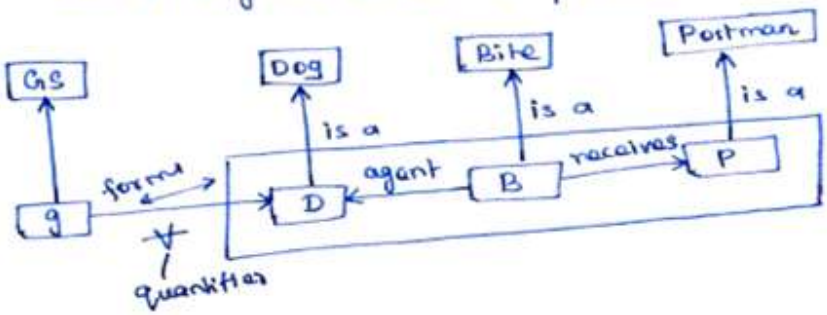
→ How to represent:

* Generate a General stmt GS. This is a special class.
* Make node 'g' as instance of GS
* Every stmts will have 2 attributes.

① A form that states which relations is being asserted

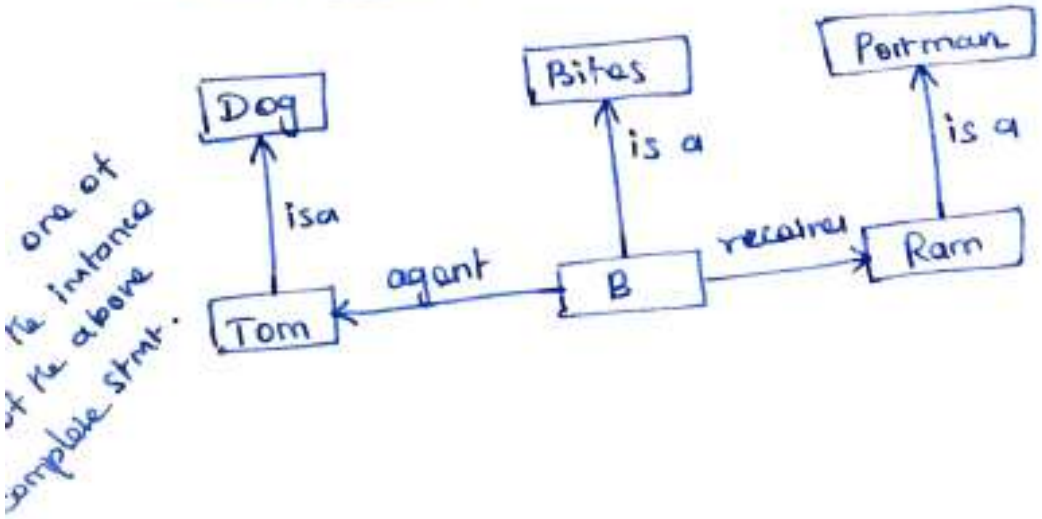
② For all '∀' and ∃ connections, that represent universally quantified variables and sentence.

Eg Every dog has bitten a postman.



Spacific

The dog Tom has bitten the postman Ram.



ore of
he instance
f he above
omplete stmt.

Frames:

stud
name
year
reg.no
mark

* Natural Language understanding require inference (ie) assumptions about what is typically true of the objects or situations under consideration, such information is coded into structures known as frames.

* A frame is a collection of attributes or slots and associated values that describe some real world entity

Slot	Filler
Publisher	Pearson
Title	AI
Author	SAM

Frame for Book.

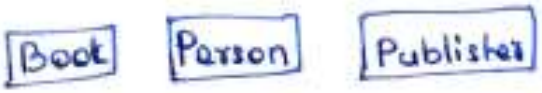
student

* A frame is similar to a record structure and corresponding to the fields and values are slot fillers.

* Procedures attached to the slots are called procedural attachments. 3 types

1) If needed
2) Default
3) If added

* Frames can be attached to another frames creating a N/W of frames.

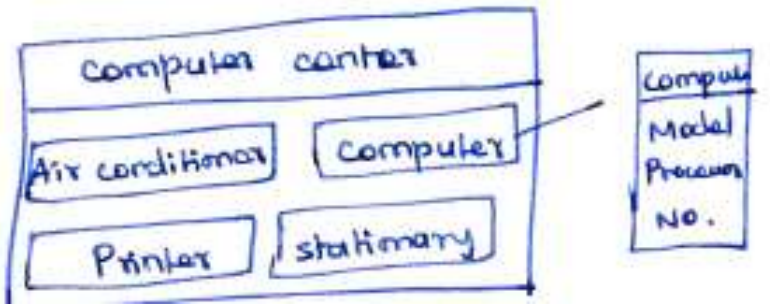


- These are all frames, and we can have different fillers for this.

Types of frames:

1) Declarative frame
   → A frame that just contains only description about objects is called declarative frames.

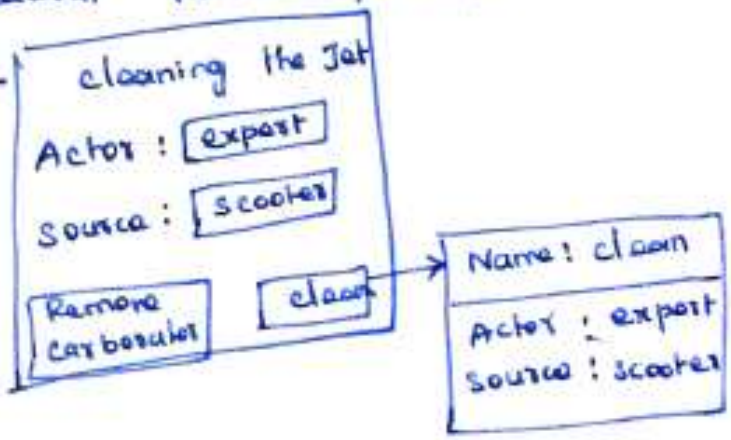
Eg

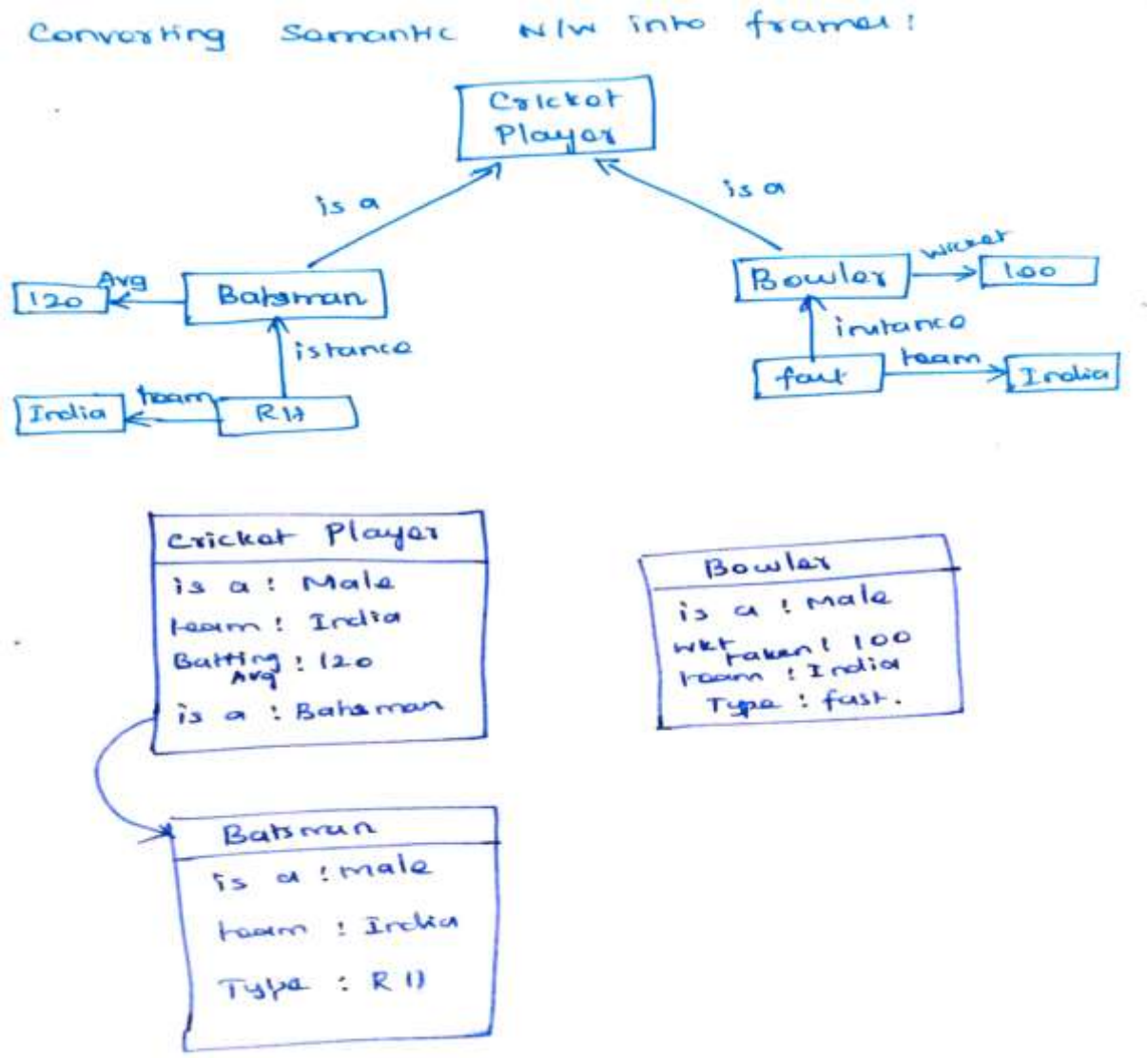


2) Procedural frame

→ It is possible to attach slots which explain how to perform things or it is possible to have procedural knowledge represented in a frame.

Eg





STRONG SLOT AND FILLER STRUCTURES

- Represent links between objects according to more *rigid* rules.
- Specific notions of what types of object and relations between them are provided.
- Represent knowledge about common situations.

Conceptual Dependency (CD)

Conceptual Dependency originally developed to represent knowledge acquired from natural language input.

The goals of this theory are:

- To help in the drawing of inference from sentences.
- To be independent of the words used in the original input.
- That is to say: *For any 2 (or more) sentences that are identical in meaning there should be only one representation of that meaning.*

It has been used by many programs that portend to understand English (*MARGIE, SAM, PAM*). CD developed by Schank *et al.* as were the previous examples.

CD provides:

- a structure into which nodes representing information can be placed
- a specific set of primitives
- at a given level of granularity.

Sentences are represented as a series of diagrams depicting actions using both abstract and real physical situations.

- The agent and the objects are represented
- The actions are built up from a set of primitive acts which can be modified by tense.

Examples of Primitive Acts are:

ATRANS

-- Transfer of an abstract relationship. *e.g. give*.

PTRANS

-- Transfer of the physical location of an object. *e.g. go.*

PROPEL

-- Application of a physical force to an object. *e.g. push*.

MTRANS

-- Transfer of mental information. *e.g. tell*.

MBUILD

-- Construct new information from old. *e.g. decide*.

SPEAK

-- Utter a sound. *e.g. say*.

ATTEND

-- Focus a sense on a stimulus. *e.g. listen, watch*.

MOVE

-- Movement of a body part by owner. *e.g. punch, kick*.

GRASP

-- Actor grasping an object. *e.g. clutch*.

INGEST

-- Actor ingesting an object. *e.g. eat*.

EXPEL

-- Actor getting rid of an object from body. *e.g. ????.*

Six primitive conceptual categories provide *building blocks* which are the set of allowable dependencies in the concepts in a sentence:

PP

-- Real world objects.

ACT

-- Real world actions.

PA

-- Attributes of objects.

AA

-- Attributes of actions.

T

-- Times.

LOC

-- Locations.

Arrows indicate the direction of dependency. Letters above indicate certain relationships:

o
-- object.

R

-- recipient-donor.

I

-- instrument *e.g. eat with a spoon*.

D

-- destination *e.g. going home.*

- Double arrows (⇔) indicate *two-way* links between the actor (PP) and action (ACT).
- The actions are built from the set of primitive acts (see above).
    - These can be modified by *tense etc.*

      The use of tense and mood in describing events is extremely important and schank introduced the following modifiers:

      p

-- past

      f

-- future

t

-- transition

$\mathbf{t_s}$

-- start transition

$\mathbf{t_f}$

-- finished transition

k

-- continuing

?

-- interrogative

/

-- negative

delta

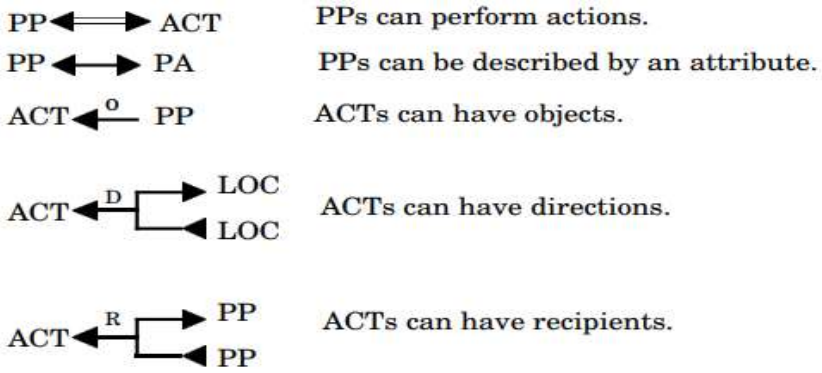
-- timeless

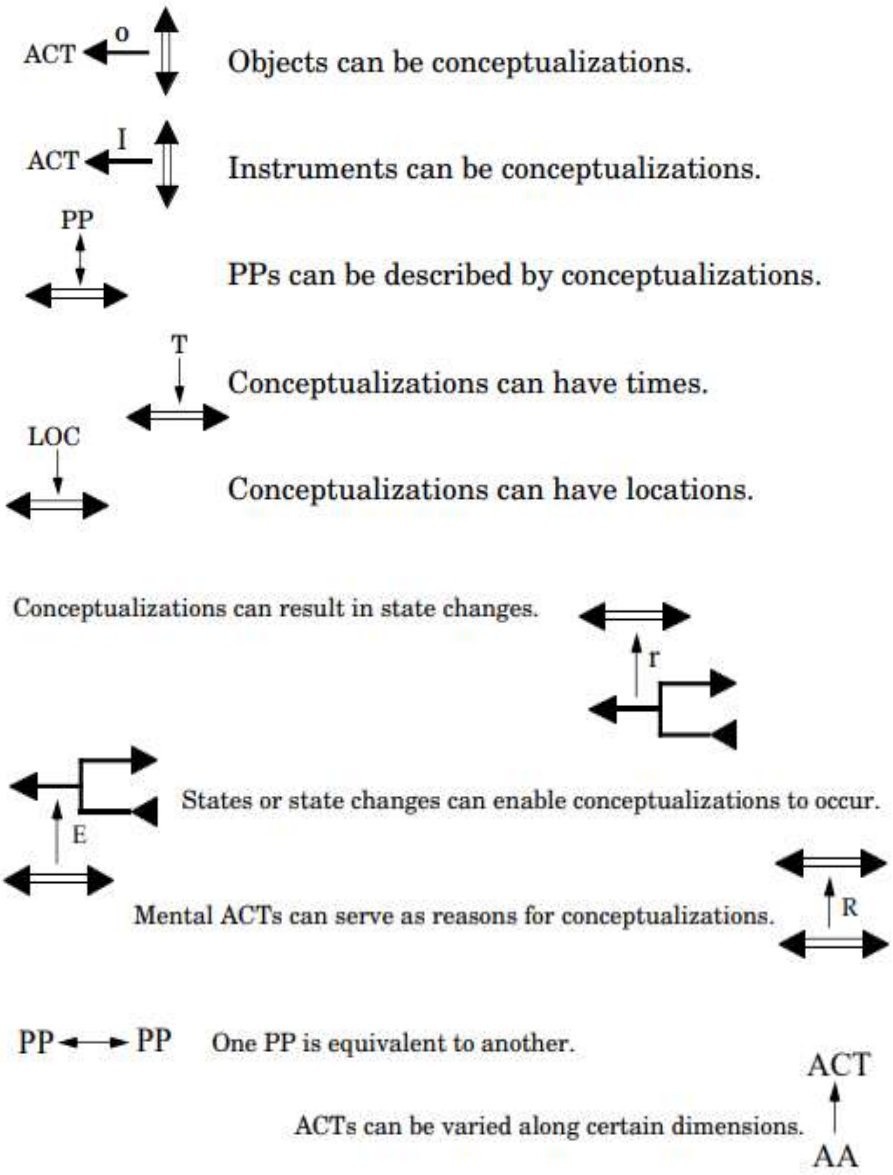
c

-- conditional

the absence of any modifier implies the *present tense*.

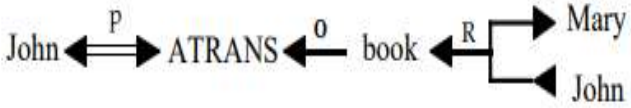
Conceptual Syntax Rules



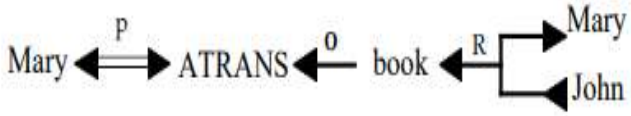


So the *past tense* of the above example:

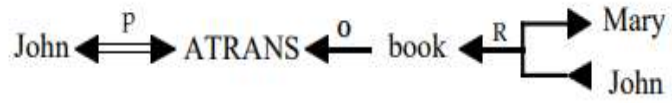
John gave Mary a book.



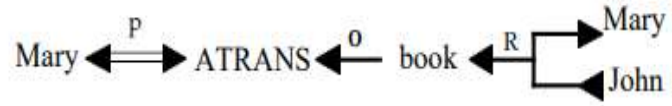
Mary took a book from John.



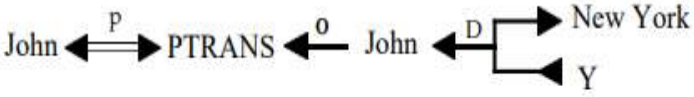
John gave Mary a book.



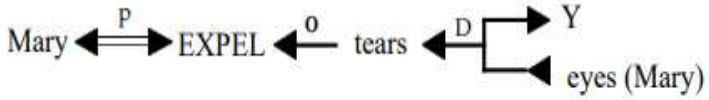
Mary took a book from John.



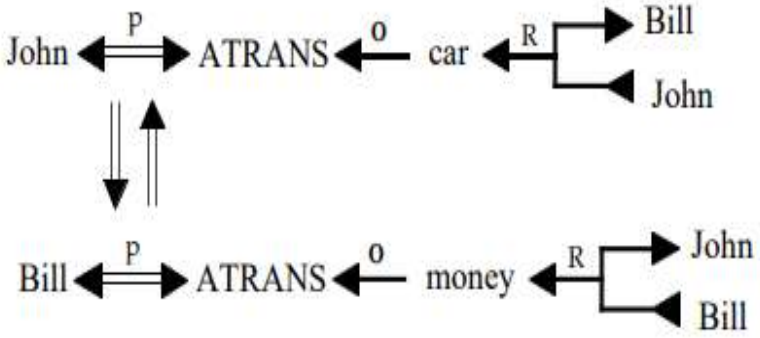
John went to New York.



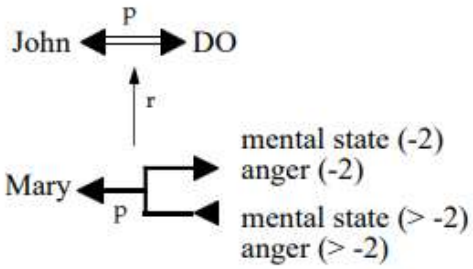
Mary cried.



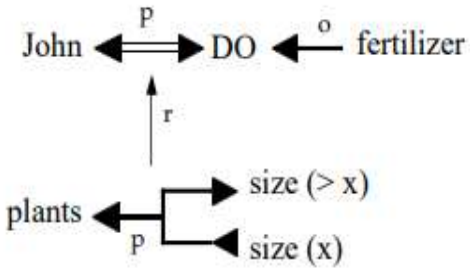
John sold his car to Bill



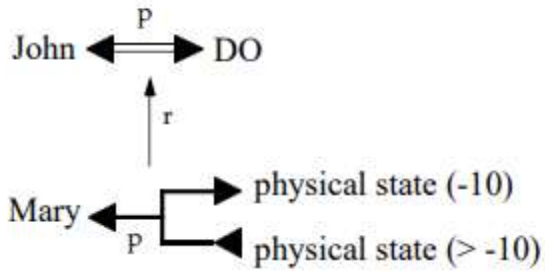
John annoyed Mary



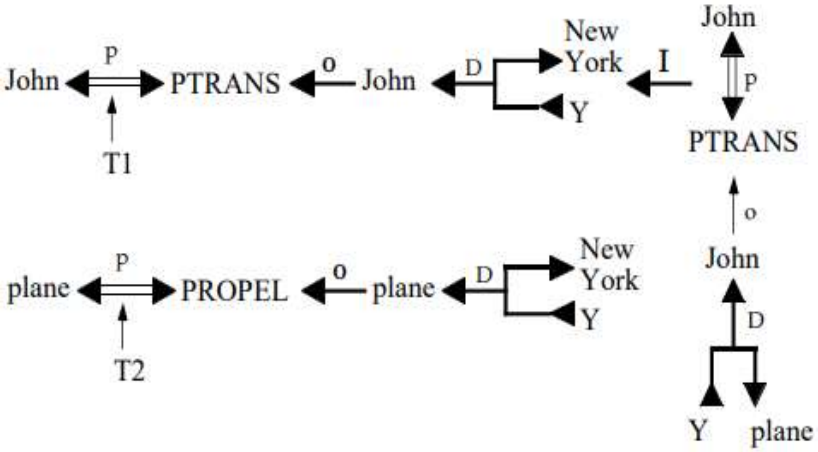
John grew the plants with fertilizer



John killed Mary



John flew to New York.



Advantages of CD:

- Using these primitives involves fewer inference rules.
- Many inference rules are already represented in CD structure.

Disadvantages of CD:

- Knowledge must be decomposed into fairly low level primitives.
- Impossible or difficult to find correct set of primitives.
- A lot of inference may still be required.
- Representations can be complex even for relatively simple actions.

Consider:

- *Dave bet Frank five pounds that Wales would win the Rugby World Cup*.
- Complex representations require a lot of storage

Applications of CD:

- MARGIE
  (*Meaning Analysis, Response Generation and Inference on English*) -- model natural language understanding.
- SAM
  (*Script Applier Mechanism*) -- Scripts to understand stories.
- PAM
  (*Plan Applier Mechanism*) -- Scripts to understand stories.

Scripts

- A *script* is a structure that prescribes a set of circumstances which could be expected to follow on from one another.
- It is similar to a thought sequence or a chain of situations which could be anticipated.
- It could be considered to consist of a number of slots or frames but with more specialised roles.

Scripts are beneficial because:

- Events tend to occur in known runs or patterns.
- Causal relationships between events exist.
- Entry conditions exist which allow an event to take place
- Prerequisites exist upon events taking place. *E.g.* when a student progresses through a degree scheme or when a purchaser buys a house.

The components of a script include:

- Entry Conditions
  -- these must be satisfied before events in the script can occur.
- Results
  -- Conditions that will be true after events in script occur.
- Props
  -- Slots representing objects involved in events.

- Roles
  -- Persons involved in the events.
- Track
  -- Variations on the script. Different tracks may share components of the same script.
- Scenes
  -- The sequence of *events* that occur. *Events* are represented in *conceptual dependency* form.

Scripts are useful in describing certain situations such as robbing a bank. This might involve:

- **Getting a gun.**
- **Hold up a bank.**
- **Escape with the money.**

Here the *Props* might be

- **Gun, *G*.**
- **Loot, *L*.**
- **Bag, *B***
- **Get away car, *C*.**

The *Roles* might be:

- **Robber, *S*.**
- **Cashier, *M*.**
- **Bank Manager, *O*.**
- **Policeman, *P*.**

The *Entry Conditions* might be:

- ***S* is poor.**
- ***S* is destitute.**

The *Results* might be:

- ***S* has more money.**
- ***O* is angry.**
- ***M* is in a state of shock.**
- ***P* is shot.**

There are 3 scenes: obtaining the gun, robbing the bank and the getaway.

Script: ROBBERY	*Track: Successful Snatch*
Props: G = Gun, L = Loot, B= Bag, C = Get away car.	*Roles*: R = Robber, M = Cashier, O = Bank Manager, P = Policeman.
Entry Conditions: R is poor. R is destitute.	*Results*: R has more money. O is angry. M is in a state of shock. P is shot.
Scene 1: Getting a gun R PTRANS R into Gun Shop R MBUILD R choice of G R MTRANS choice. R ATRANS buys G (go to scene 2)	
Scene 2 Holding up the bank R PTRANS R into bank R ATTEND eyes M, O and P R MOVE R to M position R GRASP G R MOVE G to point to M R MTRANS "Give me the money or ELSE" to M P MTRANS "Hold it Hands Up" to R R PROPEL shoots G P INGEST bullet from G M ATRANS L to M M ATRANS L puts in bag, B M PTRANS exit O ATRANS raises the alarm (go to scene 3)	
Scene 3: The getaway M PTRANS C	

Advantages of Scripts:

- Ability to predict events.
- A single coherent interpretation may be build up from a collection of observations.

Disadvantages:

- Less general than frames.
- May not be suitable to represent all kinds of knowledge.

Game Playing

- **Area of research in AI**
- **Topic of attraction to the people**
- **It has close relation intelligence**
- **Well defined states and rules**

Kinds of Games

- **Deterministic**

- **Turn-taking**
- **2-player**
- **Zero-sum**
- **Perfect information**

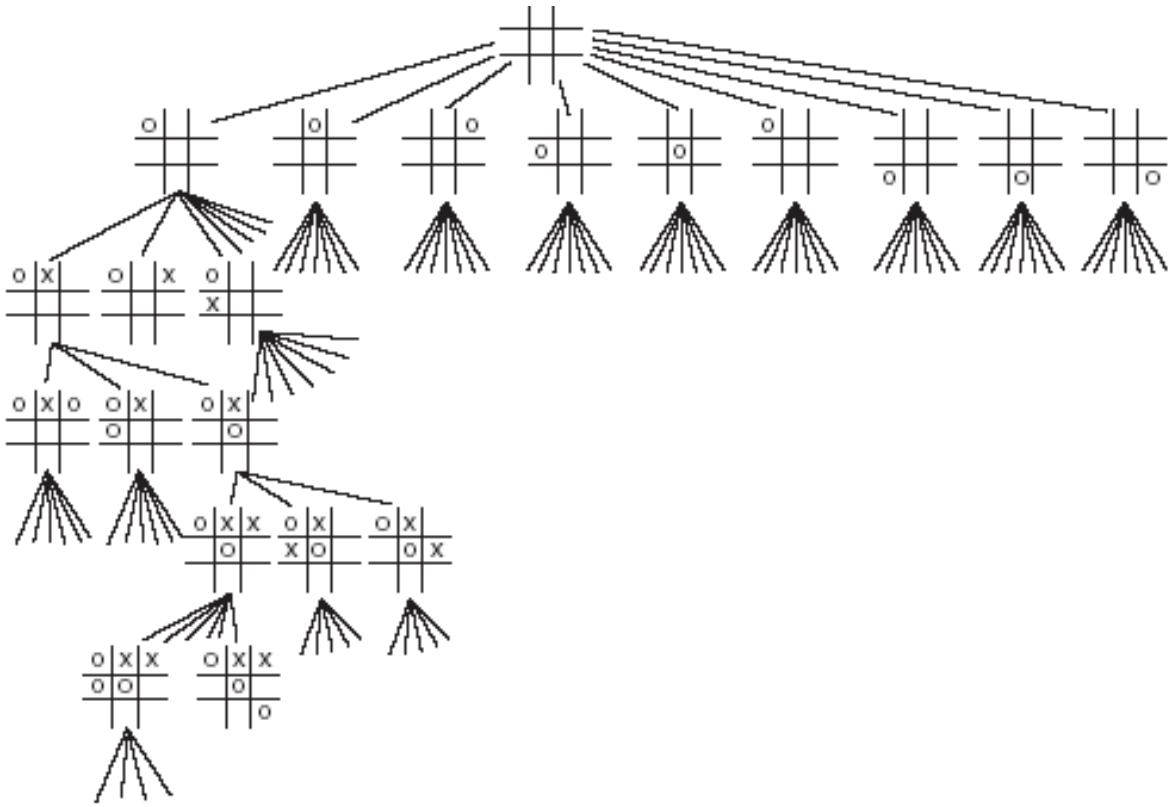
Game as Search Problem

- **Initial State: board position and player to move**
- **Successor Function: returns a list of legal (*move, state*) pairs**
- **Terminal Test: determines when the game is over**
- **Utility function: Gives a numeric value for the terminal state**

Game Trees

- **A strategy defines a complete plan of action for a given player.**
- **Game trees are used to represent two-player games.**
- **Alternate moves in the game are represented by alternate levels in the tree (plies).**
- **Nodes in the tree represent positions.**
- **Edges between nodes represent moves.**
- **Leaf nodes represent won, lost or drawn positions.**
- **Layers/levels called MAX Layer and MIN Layer.**
- **A game start at root with MAX playing first and ends at the leaf layers.**
- **Levels of game tree labelled with the outcome of the game and game ends there.**
- **Given enough processing time an optimal strategy can be found for games of perfect information by enumerating *paths of a game tree. However, in practice this can only be done for small games.***

Game Trees



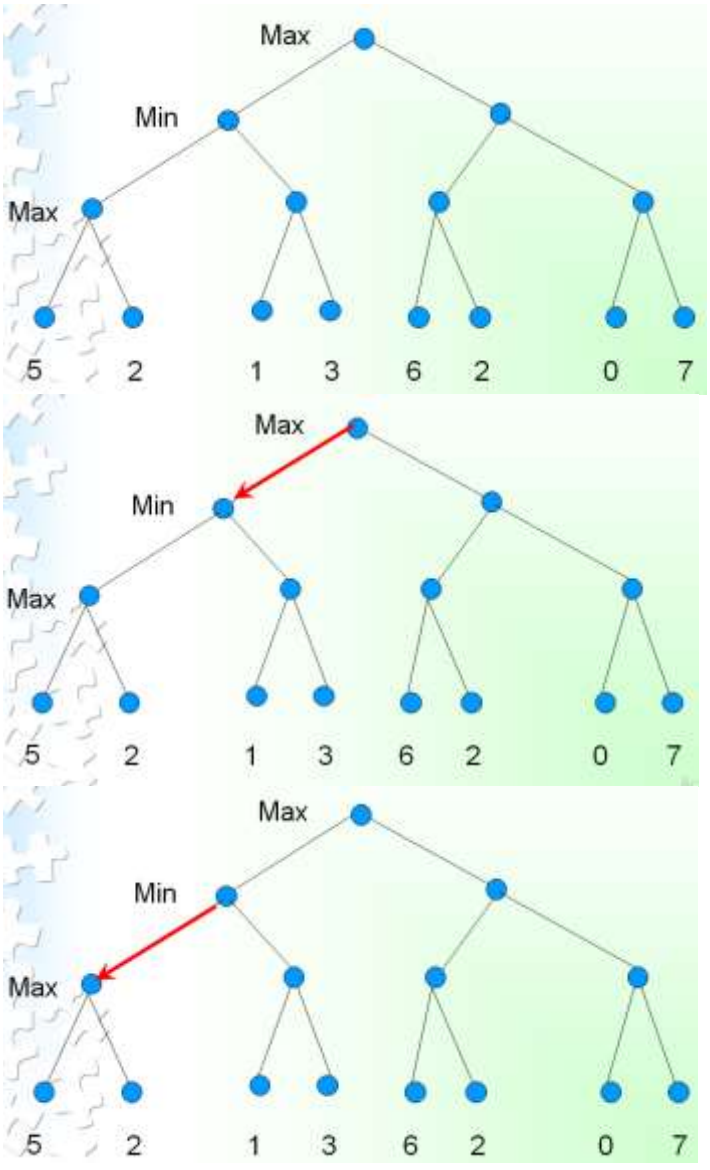
Assumptions

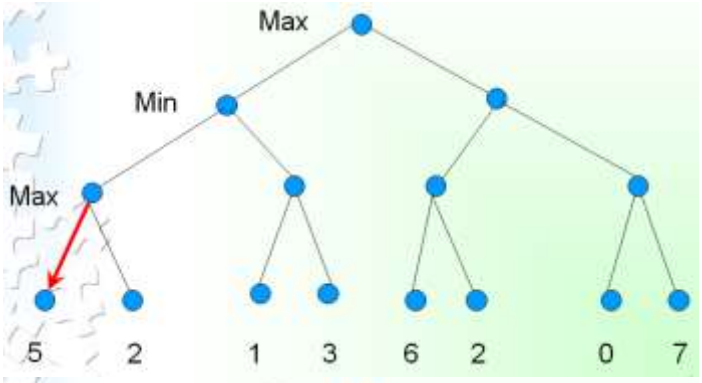
- **The most commonly used AI technique in gaming is "SEARCH"**
- **In talking about game playing systems, we make a number of assumptions:**
  - **The opponent is rational – will play to win.**
  - **The game is zero-sum – if one player wins, the other loses.**
  - **Usually, the two players have complete knowledge of the game. For games such as poker, this is clearly not true.**

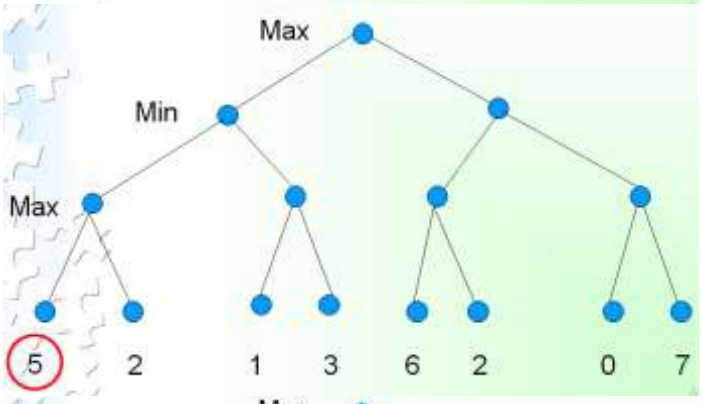
Minimax

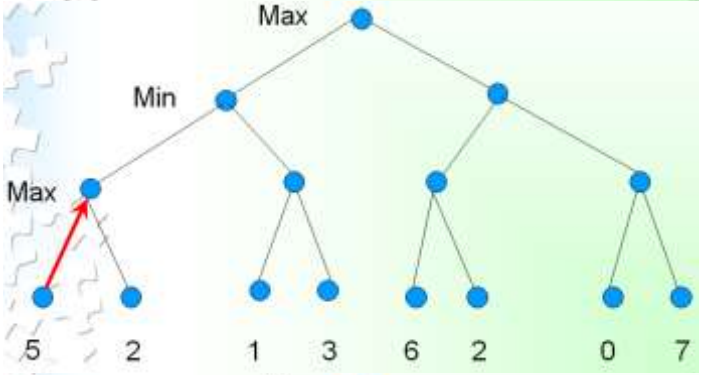
- **Minimax is a method used to evaluate game trees.**
- **A static evaluator is applied to leaf nodes, and values are passed back up the tree to determine the best score the computer can obtain against a rational opponent.**

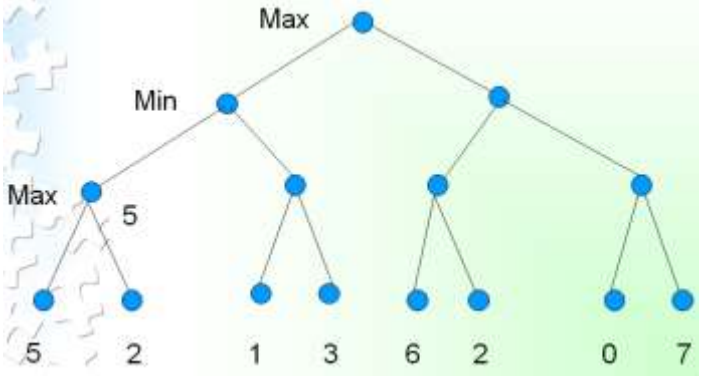
Minimax – Example

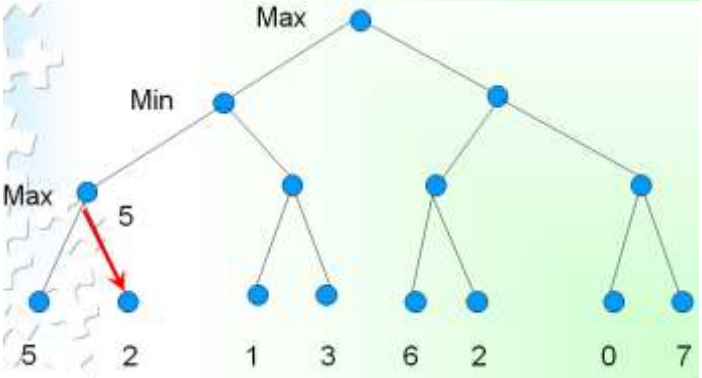


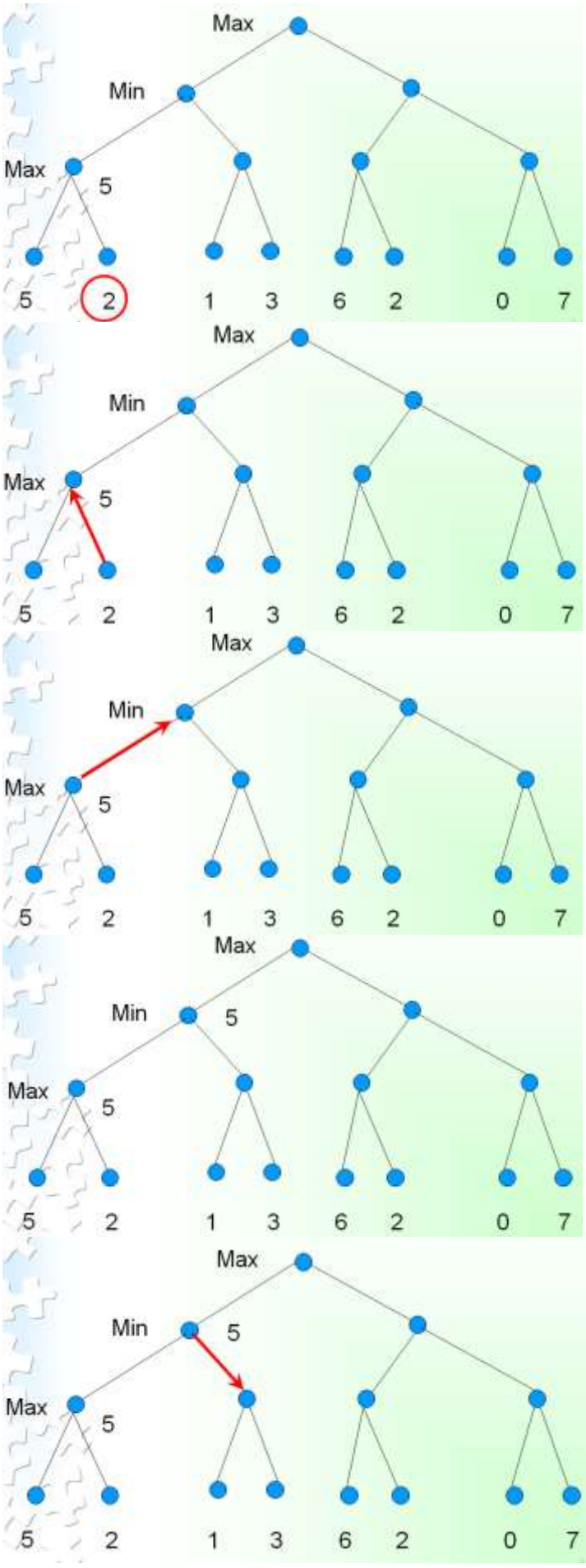

Max
Min
Max
5
2
1
3
6
2
0
7

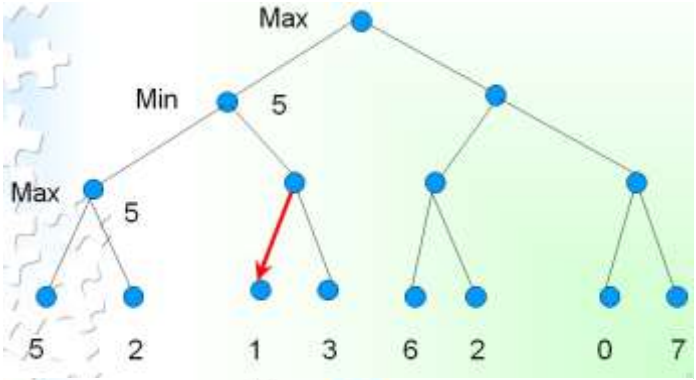

Max
Min
Max
5
2
1
3
6
2
0
7

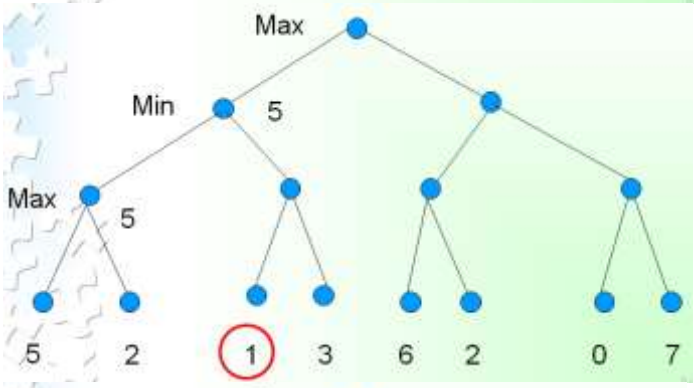

Max
Min
Max
5
2
1
3
6
2
0
7

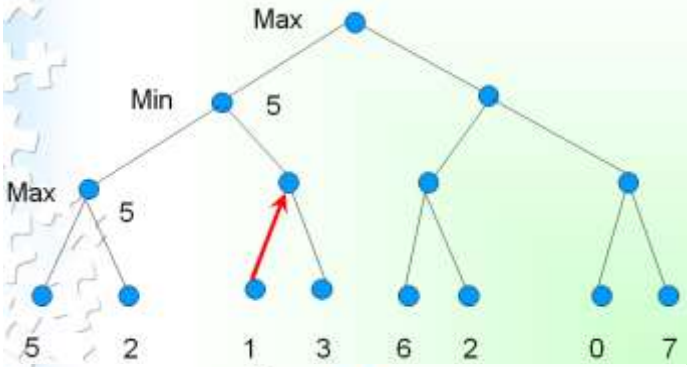

Max
Min
Max
5
5
2
1
3
6
2
0
7

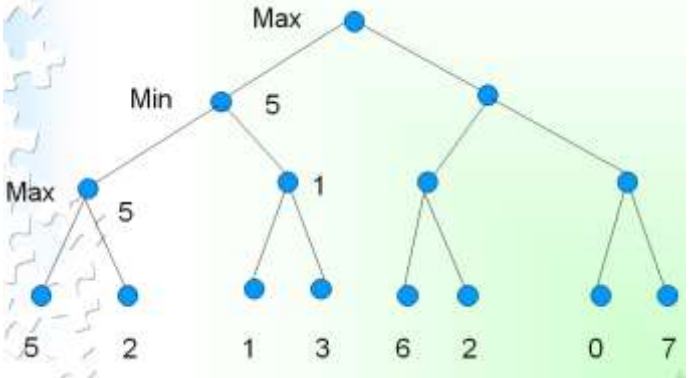

Max
Min
Max
5
5
2
1
3
6
2
0
7

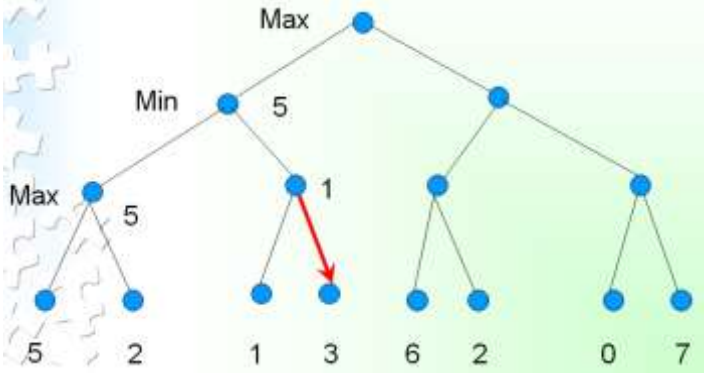

Max
Min
Max
5
5
2
1
3
6
2
0
7

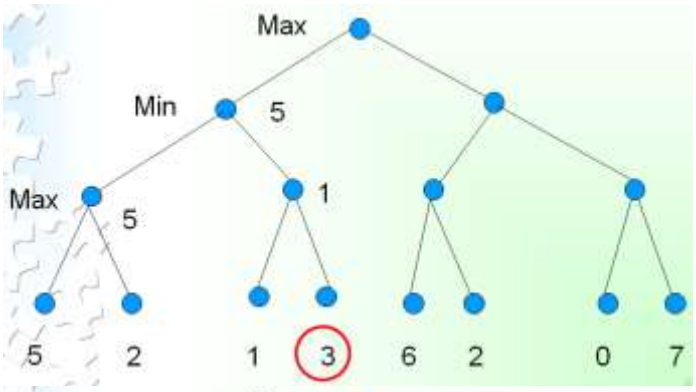

Max
Min
5
Max
5
5
2
1
3
6
2
0
7

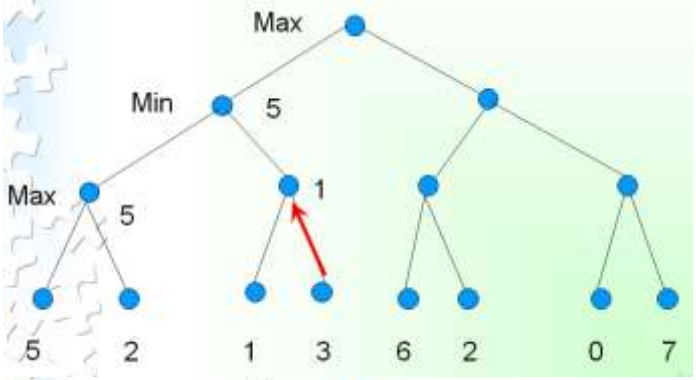

Max
Min
5
Max
5
5
2
1
3
6
2
0
7

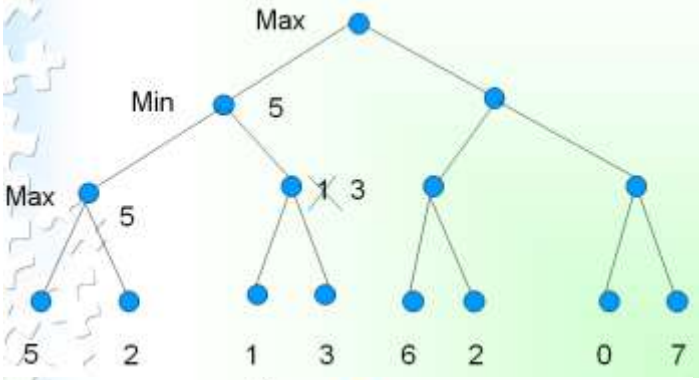

Max
Min
5
Max
5
5
2
1
3
6
2
0
7

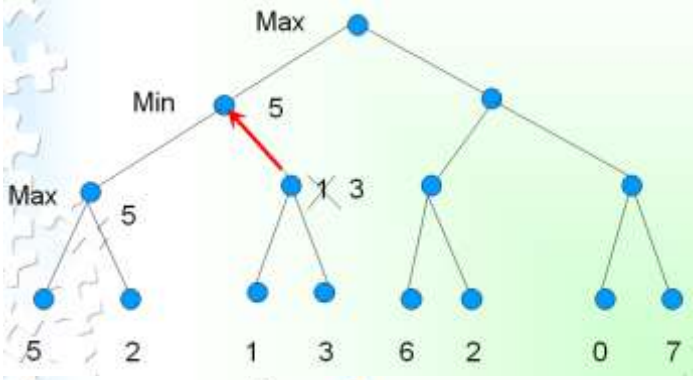

Max
Min
5
Max
1
5
5
2
1
3
6
2
0
7

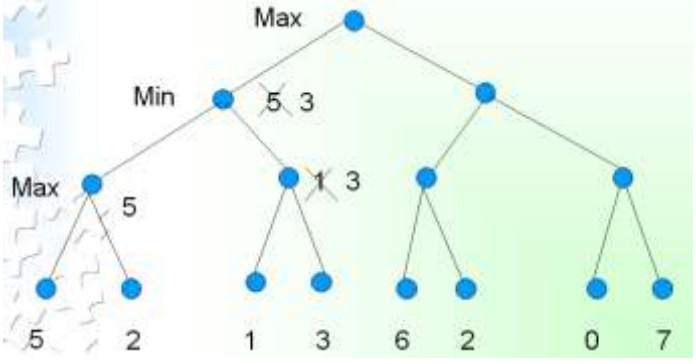

Max
Min
5
Max
1
5
5
2
1
3
6
2
0
7

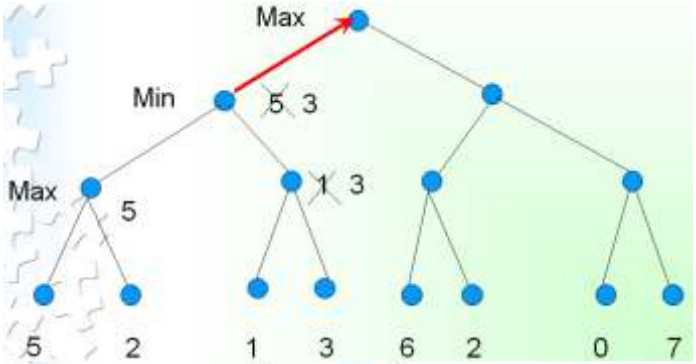

Max
Min
5
Max
5
1
5
2
1
3
6
2
0
7

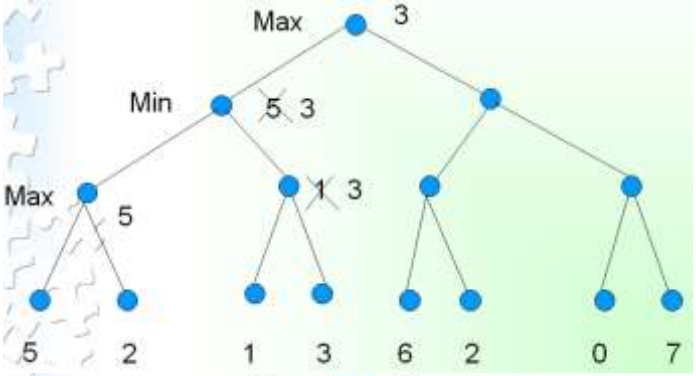

Max
Min
5
Max
5
1
5
2
1
3
6
2
0
7

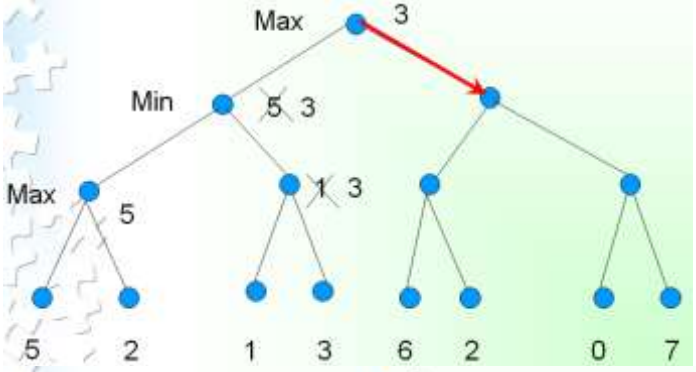

Max
Min
5
Max
5
1 3
5
2
1
3
6
2
0
7

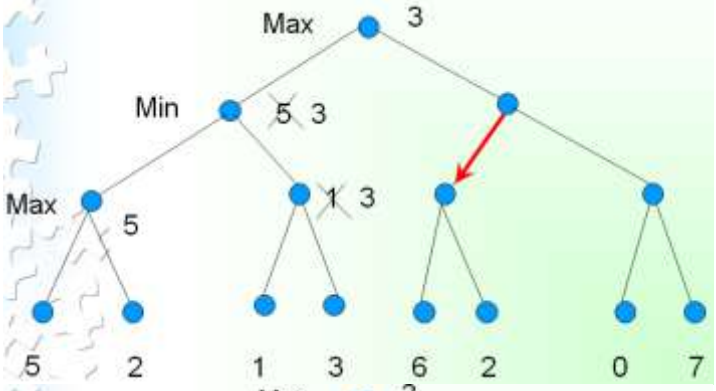

Max
Min
5
Max
5
1 3
5
2
1
3
6
2
0
7

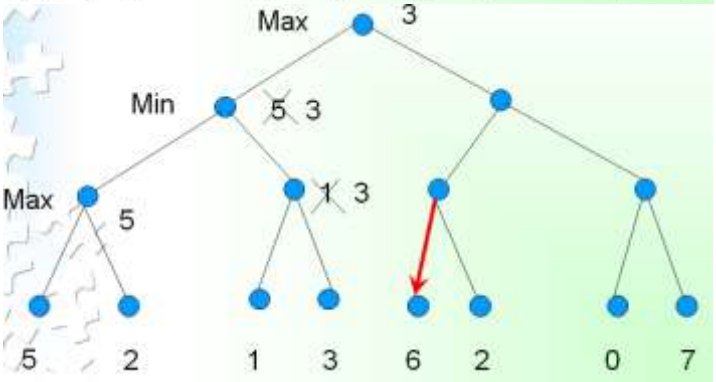

Max
Min
5 3
Max
5
1 3
5
2
1
3
6
2
0
7

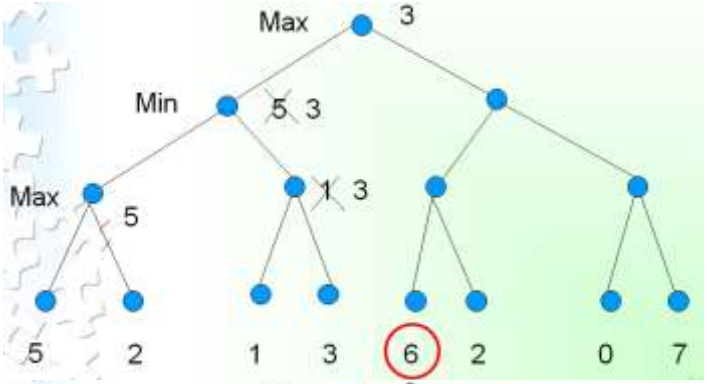

Max
Min
5 3
Max
5
1 3
5 2 1 3 6 2 0 7

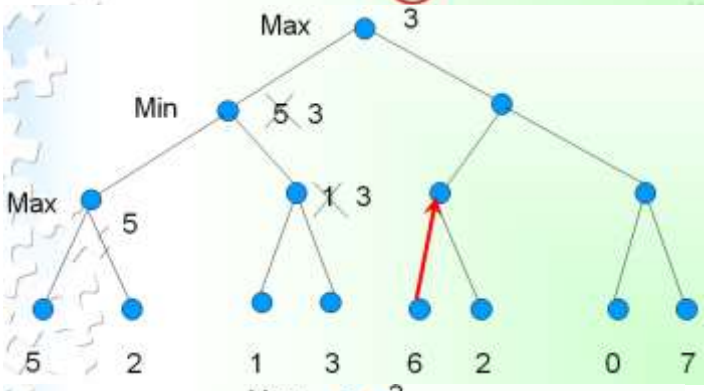

Max
3
Min
5 3
Max
5
1 3
5 2 1 3 6 2 0 7

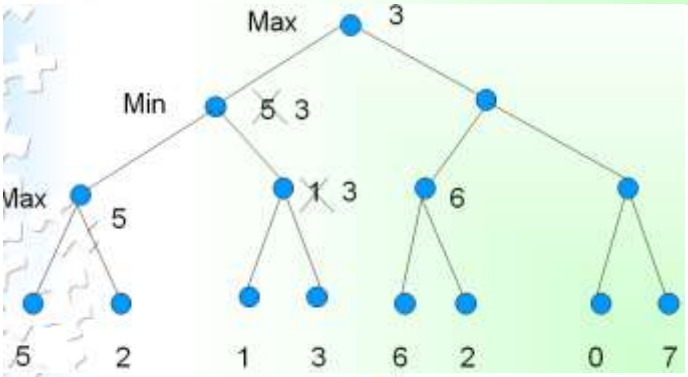

Max
3
Min
5 3
Max
5
1 3
5 2 1 3 6 2 0 7

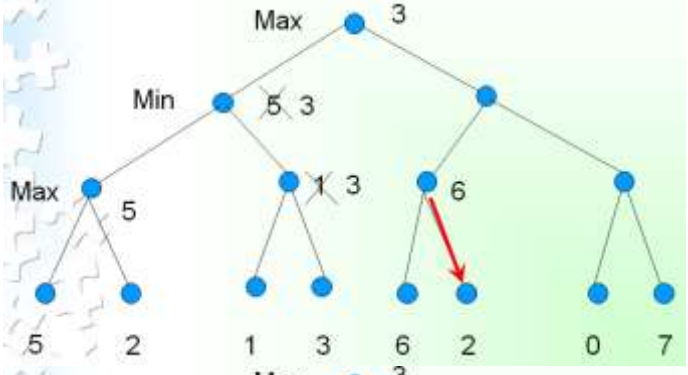

Max
3
Min
5 3
Max
5
1 3
5 2 1 3 6 2 0 7

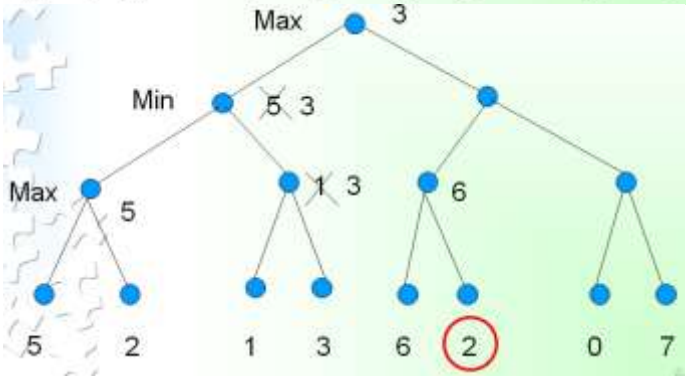

Max
3
Min
5 3
Max
5
1 3
5 2 1 3 6 2 0 7

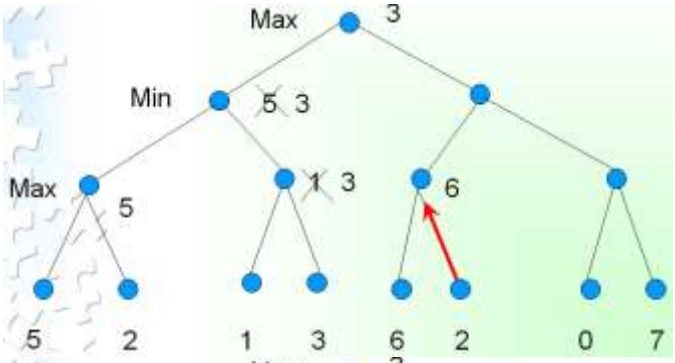

Max
3
Min
5 3
Max
5
1 3
5
2
1
3
6
2
0
7

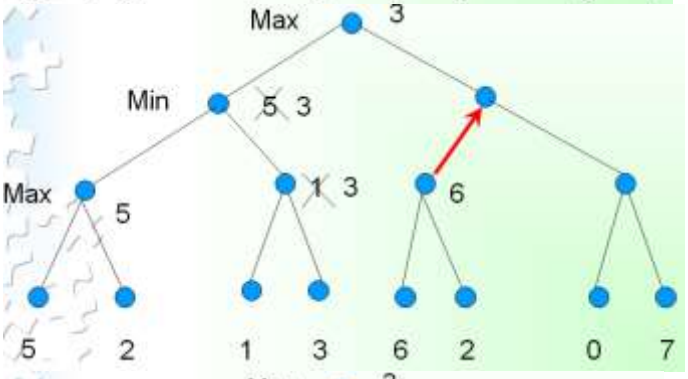

Max
3
Min
5 3
Max
5
1 3
5
2
1
3
6
2
0
7

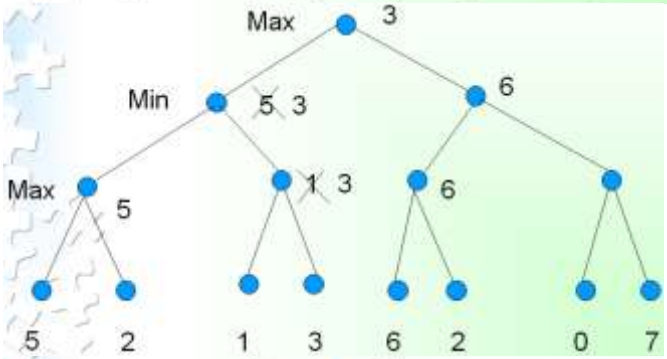

Max
3
Min
5 3
Max
5
1 3
6
5
2
1
3
6
2
0
7

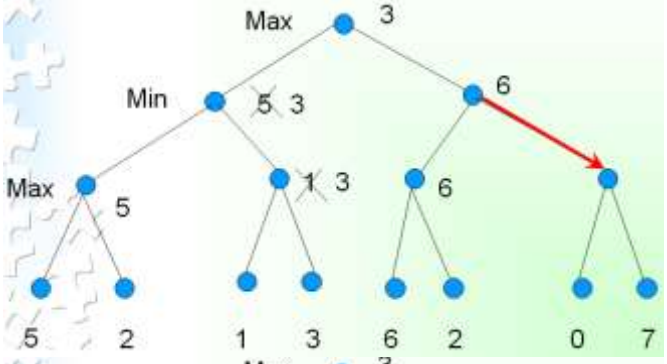

Max
3
Min
5 3
Max
5
1 3
6
5
2
1
3
6
2
0
7

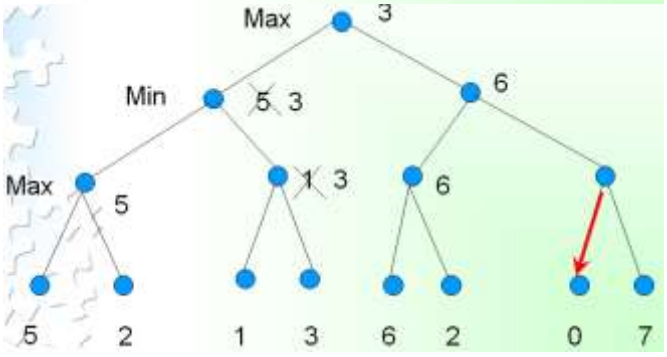

Max
3
Min
5 3
Max
5
1 3
6
5
2
1
3
6
2
0
7

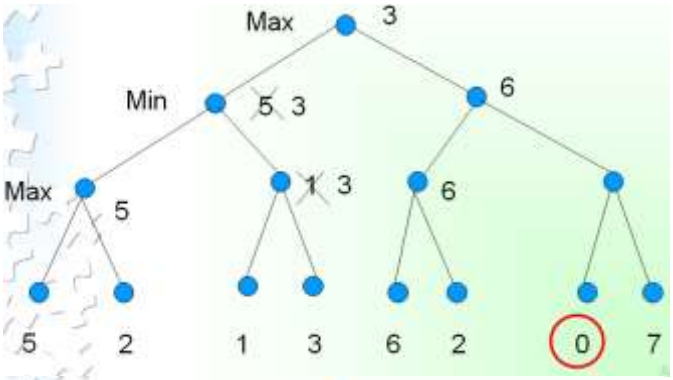

Max
3
Min
5 3
Max
5
1 3
6
5
2
1
3
6
2
0
7

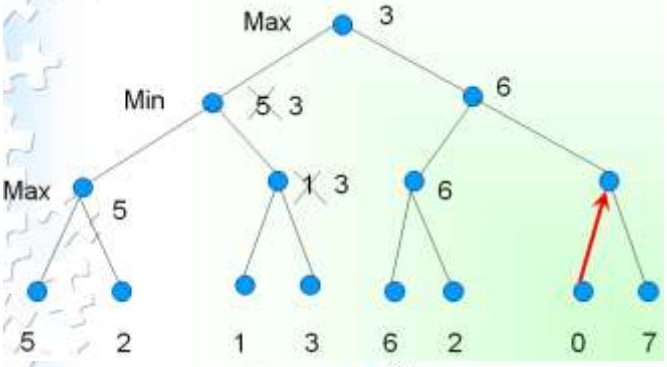

Max
3
Min
5 3
Max
5
1 3
6
5
2
1
3
6
2
0
7

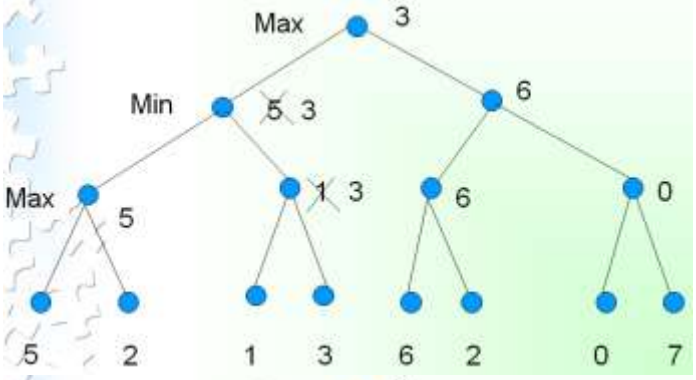

Max
3
Min
5 3
6
Max
5
1 3
6
5
2
1
3
6
2
0
7

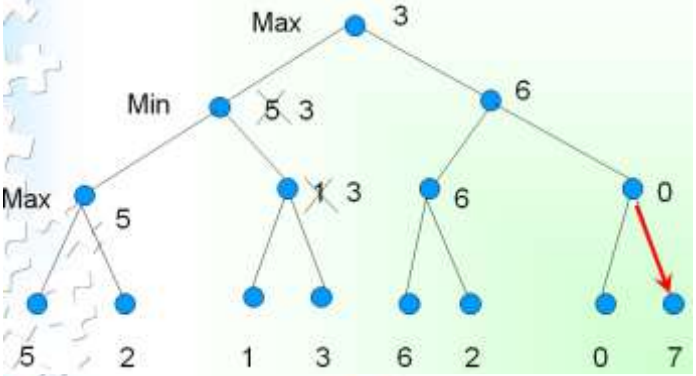

Max
3
Min
5 3
6
Max
5
1 3
6
5
2
1
3
6
2
0
7

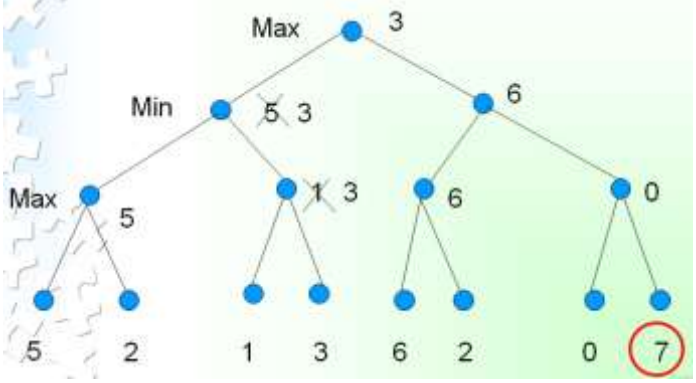

Max
3
Min
5 3
6
Max
5
1 3
6
5
2
1
3
6
2
0
7

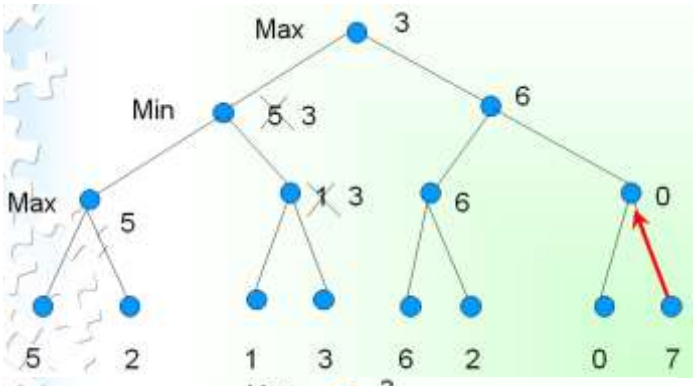

Max
3
Min
5 3
6
Max
5
1 3
6
5 2 1 3 6 2 0 7

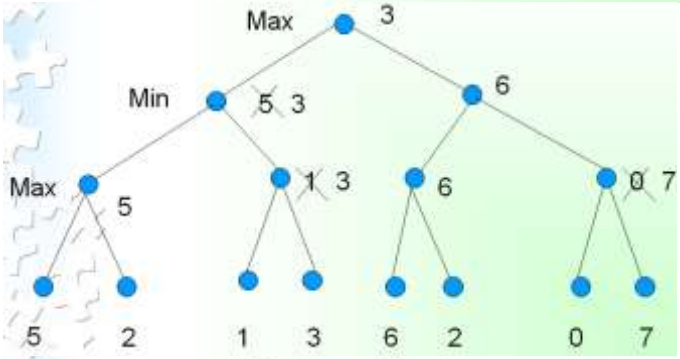

Max
3
Min
5 3
6
Max
5
1 3
6
5 2 1 3 6 2 0 7

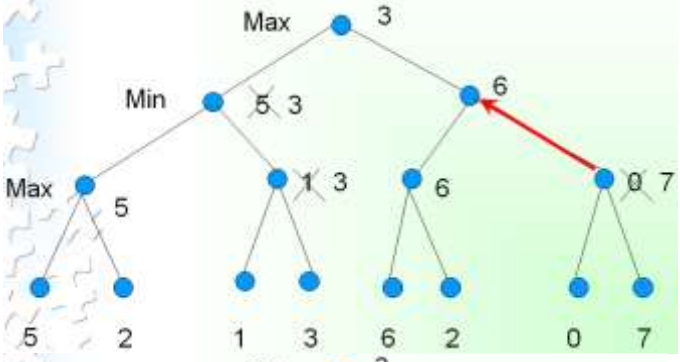

Max
3
Min
5 3
6
Max
5
1 3
6
0
5 2 1 3 6 2 0 7

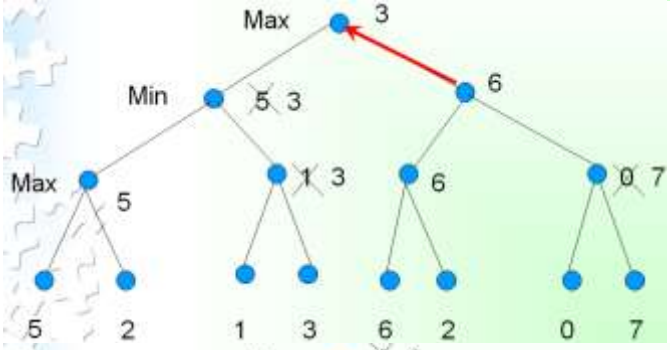

Max
3
Min
5 3
6
Max
5
1 3
6
0
5 2 1 3 6 2 0 7

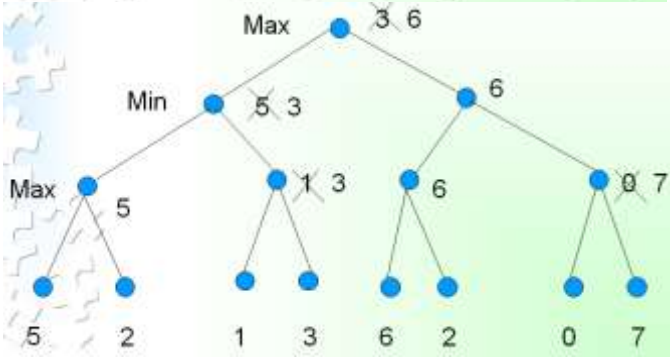

Max
3
Min
5 3
6
Max
5
1 3
6
0
5 2 1 3 6 2 0 7

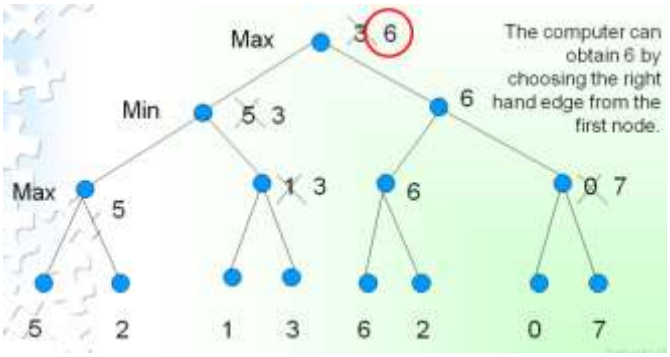

Max
3
Min
5 3
6
Max
5
1 3
6
0
5
2
1
3
6
2
0
7


Max
3
Min
5 3
6
Max
5
1 3
6
0 7
5
2
1
3
6
2
0
7


Max
3
Min
5 3
6
Max
5
1 3
6
0 7
5
2
1
3
6
2
0
7


Max
3
Min
5 3
6
Max
5
1 3
6
0 7
5
2
1
3
6
2
0
7


Max
3 6
Min
5 3
6
Max
5
1 3
6
0 7
5
2
1
3
6
2
0
7



Minimax Function

- **MINIMAX-VALUE(*n*) =**
  - **UTILITY(*n*)** **if *n* is a terminal state**
  - **$\max_{s \in Successors(n)}$ MINIMAX-VALUE(s)** **if *n* is a MAX node**
  - **$\min_{s \in Successors(n)}$ MINIMAX-VALUE(*s*)** **if *n* is a MIN node**

Searching Game Trees

- **Exhaustively searching a game tree is not usually a good idea.**
- **Even for a game as simple as tic-tac-toe there are over 350,000 nodes in the complete game tree.**
- **An additional problem is that the computer only gets to choose every other path through the tree – the opponent chooses the others.**

Alpha-beta Pruning

- **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.**

Condition for Alpha-beta pruning:

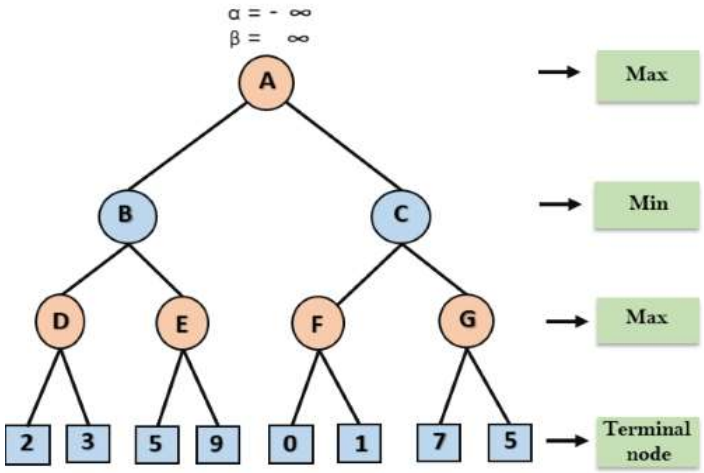
- **The main condition which required for alpha-beta pruning is:**

$$\alpha >= \beta$$

Key points about alpha-beta pruning:

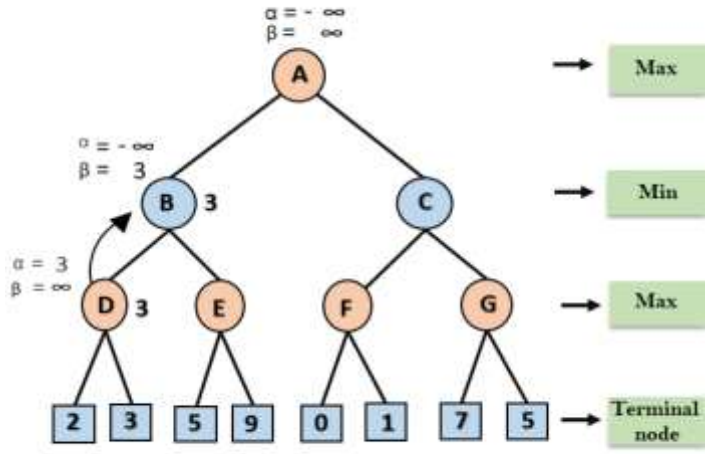
- **The Max player will only update the value of alpha.**
- **The Min player will only update the value of beta.**
- **While backtracking the tree, the node values will be passed to upper nodes instead of values of alpha and beta.**
- **We will only pass the alpha, beta values to the child nodes.**

Alpha-beta Pruning – Example. Step 1: At the first step the, Max player will start first move from node A where α= -∞ and β= +∞, these value of alpha and beta passed down to node B where again α= -∞ and β= +∞, and Node B passes the same value to its child D.



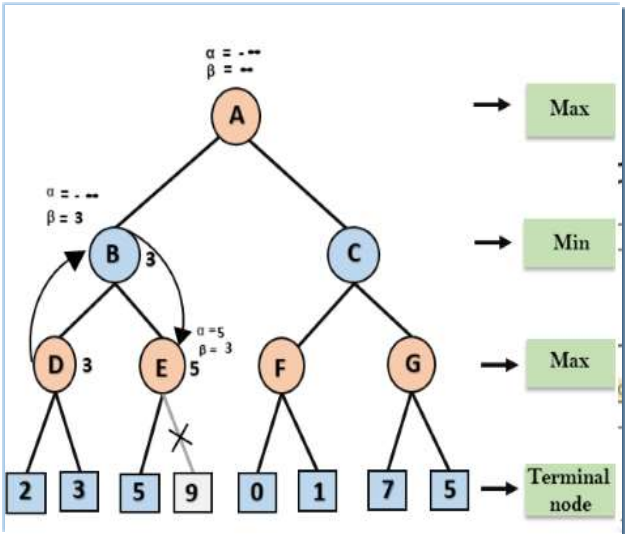
Step 2: At Node D, the value of α will be calculated as its turn for Max. The value of α is compared with firstly 2 and then 3, and the max (2, 3) = 3 will be the value of α at node D and node value will also 3.

Step 3: Now algorithm backtrack to node B, where the value of β will change as this is a turn of Min, Now β= +∞, will compare with the available subsequent nodes value, i.e. min (∞, 3) = 3, hence at node B now α= -∞, and β= 3.

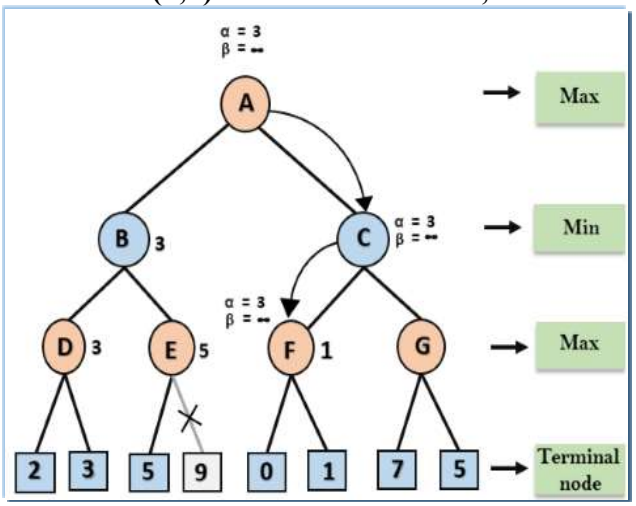


In the next step, algorithm traverse the next successor of Node B which is node E, and the values of α= -∞, and β= 3 will also be passed.

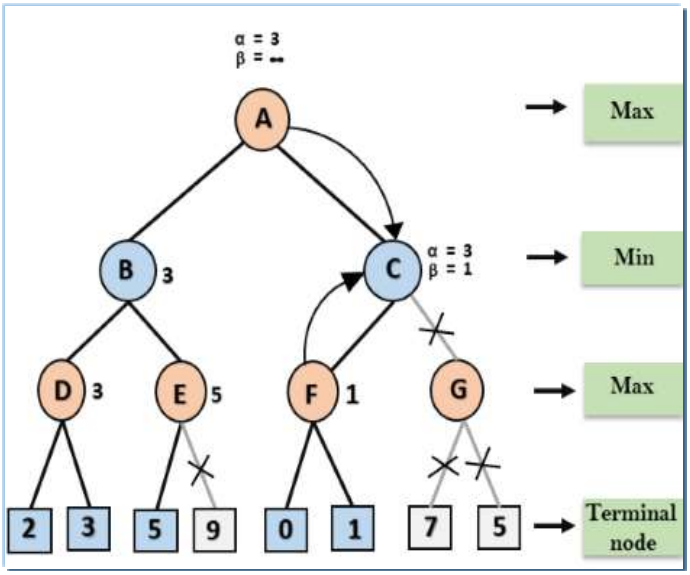
Step 4: At node E, Max will take its turn, and the value of alpha will change. The current value of alpha will be compared with 5, so max (-∞, 5) = 5, hence at node E α= 5 and β= 3, where α>=β, so the right successor of E will be pruned, and algorithm will not traverse it, and the value at node E will be 5.



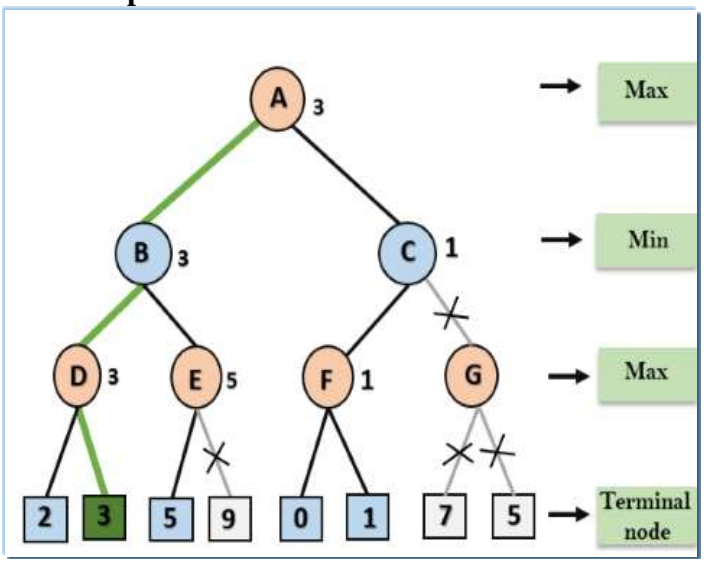
- **Step 5: At next step, algorithm again backtrack the tree, from node B to node A. At node A, the value of alpha will be changed the maximum available value is 3 as max (-∞, 3)= 3, and β= +∞, these two values now passes to right successor of A which is Node C.**
- **At node C, α=3 and β= +∞, and the same values will be passed on to node F.**
- **Step 6: At node F, again the value of α will be compared with left child which is 0, and max(3,0)= 3, and then compared with right child which is 1, and max(3,1)= 3 still α remains 3, but the node value of F will become 1.**



Step 7: Node F returns the node value 1 to node C, at C α= 3 and β= +∞, here the value of beta will be changed, it will compare with 1 so min (∞, 1) = 1. Now at C, α=3 and β= 1, and again it satisfies the condition α>=β, so the next child of C which is G will be pruned, and the algorithm will not compute the entire sub-tree G.



Step 8: C now returns the value of 1 to A here the best value for A is max (3, 1) = 3. Following is the final game tree which is the showing the nodes which are computed and nodes which has never computed. Hence the optimal value for the maximizer is 3 for this example.



Rules to find good ordering:

- **Following are some rules to find good ordering in alpha-beta pruning:**
- **Occur the best move from the shallowest node.**
- **Order the nodes in the tree such that the best nodes are checked first.**
- **Use domain knowledge while finding the best move. Ex: for Chess, try order: captures first, then threats, then forward moves, backward moves.**
- **We can bookkeep the states, as there is a possibility that states may repeat.**

Bounded Lookahead

- **For trees with high depth or very high branching factor, minimax cannot be applied to the entire tree.**
- **In such cases, bounded lookahead is applied:**
  - **search is cut off at specified depth**
  - **static evaluator applied.**
- **Horizon effect**

Static Evaluation Functions

- **A static evaluator assigns a score to a position:**
  - **High positive = computer is winning**
  - **Zero = even game**
  - **High negative = opponent is winning**
- **It is most important that a static evaluator will give a better score to a better position – the actual values are not so important.**

Creating Evaluation Function

- **How likely are you to win from a given position?**
- **Usually weighted linear functions**
  - **Different scores are given to different position and added together in a weighted fashion**
- **Possible chess function**
  - $score = 9q + 5r + 3b + 3n + p$
    - **q = # of queens, r = # of rooks, b = # of bishops, n = # of knights, and p = # of pawns**

Checkers

- **In 1959, Arthur Samuel creates a computer program that could play checkers to a high level using minimax and alpha-beta pruning.**
- **Chinook, developed in Canada defeated the world champion:**
  - **Uses alpha-beta pruning.**
  - **Has a database of millions of end games.**
  - **Also has a database of openings.**
  - **Uses heuristics and knowledge about the game.**

Chess

- **In 1997, Deep Blue defeated world champion, Garry Kasparov.**
- **This has not yet been repeated.**
- **Current systems use parallel search, alpha-beta pruning, databases of openings and heuristics.**
- **The deeper in a search tree the computer can search, the better it plays.**

Go

- **Go is a complex game played on a 19x19 board.**
- **Average branching factor in search tree around 360 (compared to 38 for chess).**
- **The best computer programs cannot compete yet with the best human players.**
- **Methods use pattern matching or selective search to explore the most appropriate parts of the search tree.**

Games of Chance

- **The methods described so far do not work well with games of chance such as poker or backgammon.**
- **Expectiminimax is a variant of minimax designed to deal with chance.**

- **Nodes have expected values based on probabilities.**

What is planning in AI?

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

Blocks-World planning problem

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

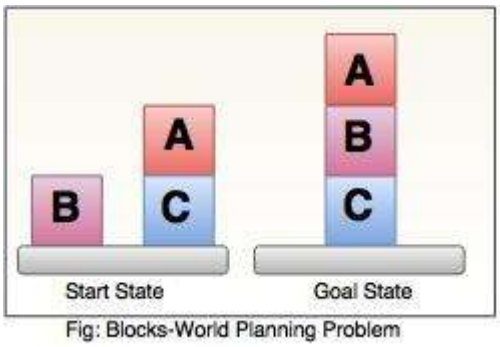


Fig: Blocks-World Planning Problem

Components of Planning System

The planning consists of following important steps:

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

Basic Components of a Planning System

When a particular problem will be solved, at that time some specific rules regarding to that problem are to be applied. Then apply the choosen rule to compute the new problem state that arises from its application. Detect when a solution has been found and calculate the active and inactive ends of that problem. Various components of a planning system are described as follows.

(a) **States:** For a planning process, the planners decompose the world into some environments. Then environments are defined by some logical conditions and states. The problems can be viewed as the task of finding a path from a given starting state to

some desirable goal state. The state can be viewed as a conjunction of positive literals. For example, Rich A famous might represent the state of a best agent.

(b)**Goal:** A goal is a specified state. To find a solution to a problem using a search procedure is to generate moves through the problem space until a goal state is reached. In the context of game playing programs, a goal state is one in which we win. Unfortunately, for interesting games like chess, it is not usually, possible, even with a good plausible move generator, to search until a goal state is found.

(c) **Actions:** An action is specified en terms of the pre-conditions that must hold before it can be executed and then the effects that ensue when it is executed. For example, an action for running a **Action**tigerfrom**Run**one**T,**locationfrom,to**,**another is

Action (Run (T, from, to),

PRECONDITION: At (T, from) ∧ Tiger (T) ∧ Jungle (from) ∧ Jungle (To)

EFFECT: ~ At (T, from) ∧ At (T, to))

(d)**Precondition:** The precondition is a conjunction of function free positive literals stating what must be true in a state before the action can be executed.

(e) **Effect:** It is a conjunction of function free literals describing how the state changes when the action is executed.

(f) **Finding a solution:** A planning system has succeeded in finding a solution to a problem when it has found a sequence of operators that transforms the initial problem state into the goal state. The way it can be solved depends on the way that state descriptions are represented.

(g) **Calculating the Dead State:** As a planning system is searching for a sequence of operators to solve a particular problem, it must be able to detect when it is exploring a path that can never lead to a solution. The same reasoning methods that can be used to detect a solution can often be used for detecting a dead path. If the search process is reasoning in forward direction from the initial state, it can prune any path that leads to a state from which the goal state cannot be reached. If the search process is reasoning backward from the goal state, it can also terminate a path either because it is sure that the starting state cannot be reached.

ITERATIVE DEEPENING

* While still an unintelligent algorithm, the iterative deepening Search combines the Positive elements of breadth-first and depth-first Searching to Create an algorithm which is often an improvement Over each method individually.

* An Iterative Deepening Search Operates like a depth-first Search, except slightly more constrained there is a maximum depth which defines how many levels deep the algorithm can look for Solutions.

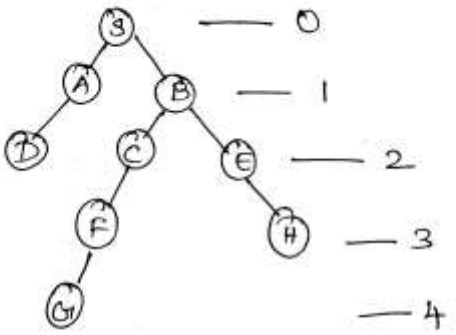
* A node at the maximum level of depth is treated as terminal, even if it would Ordinarily have Successor nodes.

* If a Search "fails", then the maximum level is increased by one and the process repeats.

* The value for the maximum depth is initially Set at 0.

goal - g
Source - S

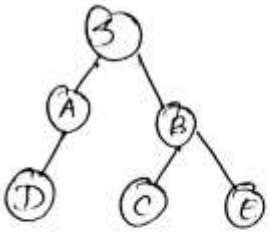
Initially
MAX = 0



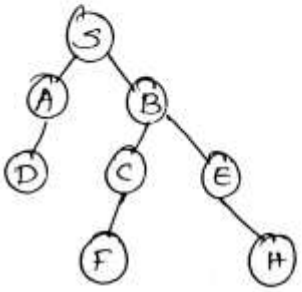
MAX = 0

MAX = 1

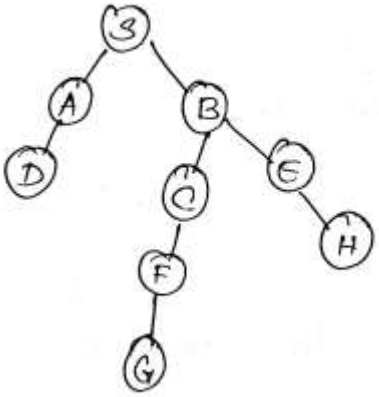
MAX - 2



MAX - 3



MAX - 4



* this continues until a solution is found.

* An Interesting Observation is that the nodes in this Search are first seen checked in the same order they would be checked in breadth-First-Search, however, since nodes are deleted as the Search progresses, much less memory is used at any given time.

The drawback to the Iterative Deepening Search is clear from the walkthrough - it can be painfully redundant, rechecking every node it has already checked with each new iteration.

Goal stack planning

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

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

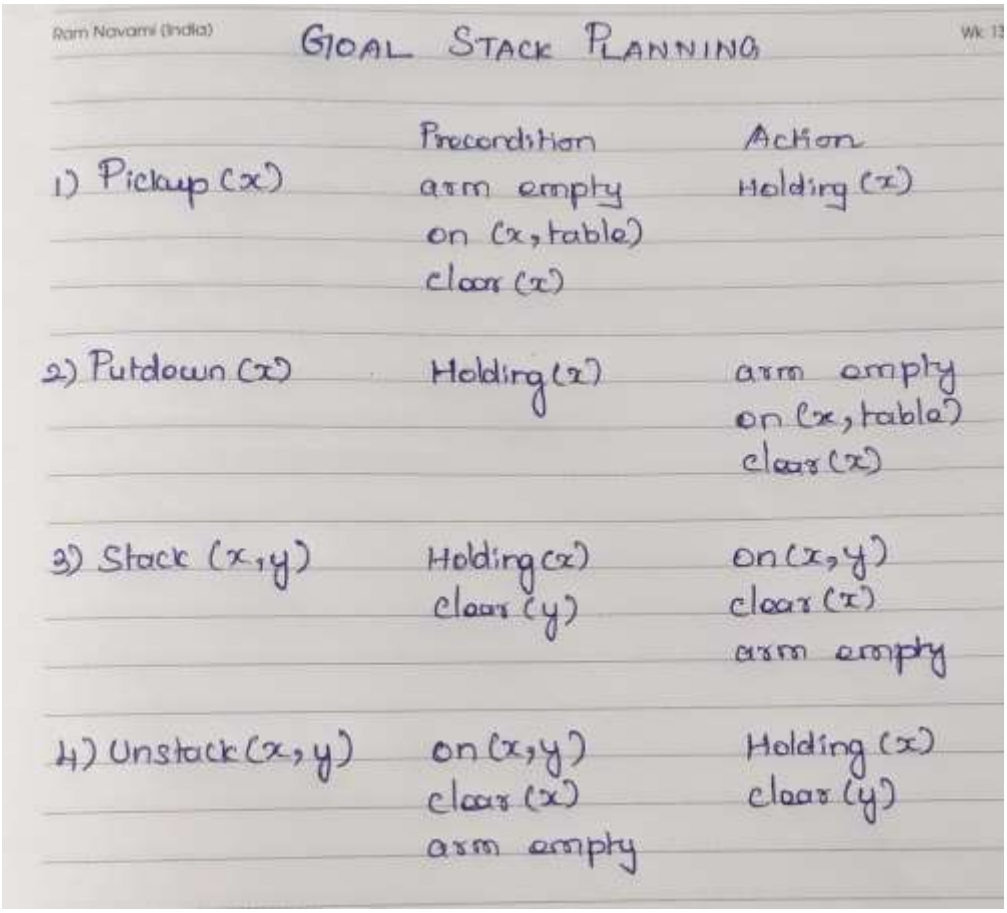
The important steps of the algorithm are as stated below:

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 subgoals 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.

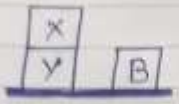
iii. If stack top is an action, pop it from the stack, execute it and change the knowledge base by the effects of the action.

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



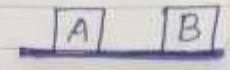
Mar' 18

Example

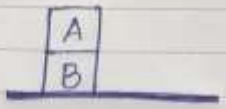


arm

Start State :



Goal State :



arm

25 Mar' 18 Sunday

on (A, B)

Start State

on (A, table)
on (B, table)

Goal State

On (A, B)

on (A, B)

↓

Stack (A, B)

↓

× Holding (A) → Pickup

✓ clear (B)

↓

arm empty
on (A, table)
clear (A)