AI mod-3,4,5

1) What is knowledge-based agent? Explain it's importance in problem solving techniques.

→ The central component of a knowledge-based agent is its knowledge base, or KB. A knowledge base is a set of sentences.

Sentences are expressed using a knowledge representation language. Two generic functions :-

- TELL :- add new sentences to the KB.
- ASK :- query what is known from the KB.

-Both operations may involve inference - that is, drawing new sentences from old. The agent must be able to :

- Represent states and actions.
- Incorporate new percepts
- Update internal representation of the world.
- Deduce hidden properties of the world
- Deduce appropriate actions.

→ function KB-Agent (percept) return an action
  static : KB, a knowledge base
    t, a counter, initially 0, indicating time
  TELL(KB, MAKE-PERCEPT-SENTENCE(percept, t))
  action → ASK(KB, MAKE-ACTION-QUERY(t))
  TELL(KB, MAKE-ACTION-SENTENCE(action, t))
  t ← t+1
  return action

- Importance in problem-solving Techniques:

- Enables flexible problem-solving by adapting to new tasks and learning from its environment.
- useful in partially observable environments by reasoning over uncertain or incomplete data.
- Combines declarative knowledge representation and inference, allowing structured problem solving.
- Declarative :- You can build a KB agent simply by TELLing it what it needs to know.
- Procedural :- Encode desired behaviours directly as program code.

A successful agent combines both declarative and procedural elements in the design;

2) Write a short note on Wumpus World problem.

→ The wumpus world is a cave consisting of rooms connected by passageways. Lurking somewhere in the cave is the terrible wumpus, a beast that eats anyone who enters its room. The wumpus can be shot by an agent, but the agent has only one arrow.

Some rooms contain bottomless pits that will trap anyone who wanders into these rooms except the wumpus. The only mitigating feature of this bleak environment is the possibility of finding a heap of gold. Although the wumpus world is rather tame by modern computer game standards, it illustrates points about intelligence.

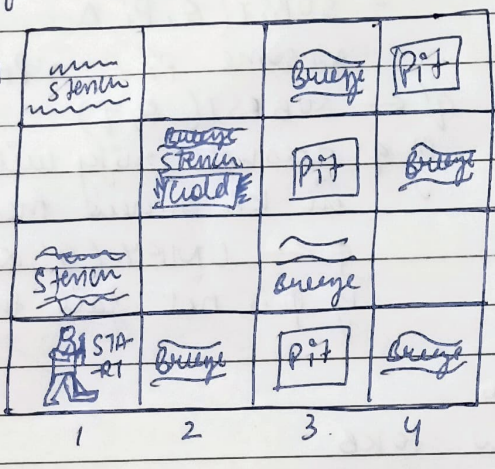
The precise definition of the game is given by the PEAS description:

- Performance measure:- +1000 for climbing out of the cave with the gold, -1000 for falling into a pit or being eaten by wumpus; -1 for each action taken and -10 for using up the arrow. The game ends when either the agent dies or climbs the cave with gold.
- Environment:- A 4×4 grid of rooms. The agent always starts in the square labeled [1,1], facing to the right. The position of gold and wumpus are chosen randomly with a uniform distribution.
- Actuators:- The agent can move Forward, TurnLeft by 90°, or TurnRight by 90°. The agent dies a miserable death if it enters a square with a pit or a live wumpus. The action 'Grab' can be used to pick up the gold if it is in the same square as the agent. The action 'Shoot' can be used to fire an arrow in a straight line in the direction the agent is facing. Finally, the action 'Climb' can be used to climb out of the cave, but only from square [1,1].
- Sensors:- The agent has five sensors, each of which gives a single bit of information:-

— In the square containing the wumpus and in the directly adjacent squares, the agent will perceive a 'Stench'.

- in the square directly adjacent to a pit, the agent will perceive a 'Breeze'.
- when the gold is present, the agent will perceive a 'glitter'.
- when an agent walks into a wall, it will perceive a 'Bump'.
- when the wumpus is killed, it emits a woeful 'scream' that can be perceived any-where in the cave.

The percepts will be given to the agent program in the form of a list of the symbols: as [Stench, Breeze, None, None, None]. example if there is only stench & breeze. It is sequential, because rewards will come only after many actions are taken. It is partially observable, because some aspects of the state are not directly perceivable: the agent's location, the wumpus's state of health, and the availability of an arrow.



A cautious agent will move only into a square that it knows to be OK.

3) Explain forward-chaining algo for Propositional definite clauses.

→ A forward chaining algorithm for propositional definite clauses was simple: start with the atomic sentences in the knowledge base and apply Modus Ponens in the forward direction, adding new atomic sentences, until no further inferences can be made.

Definite clauses such as situation ⇒ Response are especially useful for systems that make inferences in response to newly arrived information. First-order definite clauses closely resemble propositional definite clauses: they are disjunctions of literals of which exactly one is positive. A definite clause either is atomic or is

is an implication whose antecedent is a conjunction of positive literals and whose consequent is a single positive literal.

function FOL-FC-ASK(KB, α) returns a substitution or false
inputs: KB, the knowledge base, a set of first-order definite clauses;
α, the query, an atomic sentence
local variables: new, the new sentences inferred on each iteration
repeat until new is empty
new ← { }
for each rule in KB do
(P1 ∧ ... ∧ Pn ⇒ q) ← STANDARDIZE-VARIABLES(rule)
for each θ such that SUBST(θ, P1 ∧ ... ∧
= SUBST(θ, P1' ∧ ... ∧ Pn')
for some P1', ..., Pn' in KB
q' ← SUBST(θ, q)
if q' does not unify with some sentence in KB or new then add q' to new
φ ← UNIFY(q', α)
if φ is not fail then return φ
for each
add new to KB
return false

4) what is meant by FOL & give syntax & semantics.
→ First-order logic is another way of knowledge representation in artificial intelligence. It is an extension to propositional logic. FOL is sufficiently expressive to represent the natural language statements in a concise way. It is also known as Predicate logic or First-order Predicate logic.
FOL is a powerful language that develops information about the objects in a more easy way and can also express the relationship between those objects.

Syntax for FOL :-

Sentence → Atomic sentence | Complex sentence
Atomic Sentence → Predicate | Predicate (Term, ...) | Term = Term

Complex Sentence → (Sentence) | [Sentence]
| ¬ Sentence
| Sentence ∧ Sentence
| Sentence ∨ Sentence
| Sentence ⇒ Sentence
| Sentence ⇔ Sentence
| Quantifier Variable, ... Sentence

Term → Function (Term, ...)
| Constant
| Variable

Quantifier → ∀ | ∃
Constant → A | X_1 | John | ...
Variable → a | x | s | ...
Predicate → True | False | After | Loves | Raining | ...
Function → Mother | LeftLeg | ...

Operator Precedence : ¬, =, ∧, ∨, ⇒, ⇔

6) Explain steps of knowledge Engineering Projects in FOL with an example.

→

(i) Identify the task :- The knowledge engineer must delineate the range of questions that the knowledge base will support and kind of facts that will be available for each specific problem instance. For example, does the wumpus would need to be able to choose actions or is it required to answer questions only about the contents of the environment? The task will determine what knowledge must be represented.

(ii) Assemble the relevant knowledge :- The knowledge engineer might already be an expert in the domain, or might need to work with real experts to extract what they know - a process called knowledge acquisition. At this stage the knowledge is not represented formally. The idea is to understand the scope of the knowledge base, as determined

by the task. For automatic world, which is designed by an artificial set of rules, the relevant knowledge is easy to identify.

(iii) Decide on a vocabulary of predicates, functions and constants:

(iv) Encode general knowledge about the domain :-

(v) Encode a description of the specific problem instance :-

(vi) Post queries to the inference and get answers :-

(v) Debug the knowledge base :-

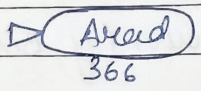
7) Explain:-
(i) State space Problem:-

8) Explain A* search / Greedy best first search strategy with an example.

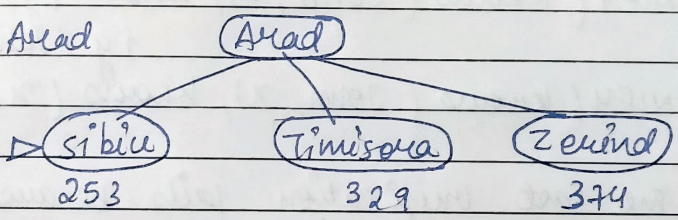
→ Greedy best first search tries to expand the node that is closest to the goal, on the grounds that this is likely to lead to a solution quickly. It evaluates nodes by using just the heuristic function; that is, $f(n) = h(n)$

For route-finding problem in Romania; we use the straight-line distance heuristic, which we will call h_{SLD}. If the goal is Bucharest, we need to know the straight-line distances to Bucharest. h_{SLD} is correlated with actual road distances and is therefore, a useful heuristic.

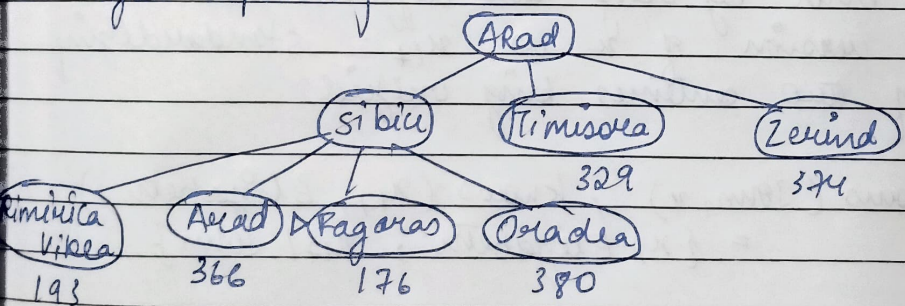
(a) initial state



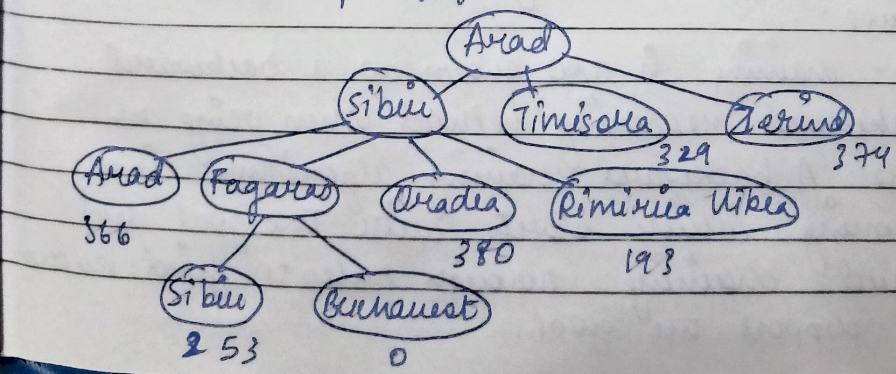
(b) After expanding Arad



(c) After expanding Sibiu



(d) after expanding fagaras



9) write a note on unification process.

→ Lifted inference rules require finding substitutions that make different logical expressions look identical. This process is called unification and is a key component of all first-order inference algorithms. The UNIFY algorithm takes two sentences and returns a unifier for them if one exists:

UNIFY (P, q) = θ where SUBST (θ, P) = SUBST (θ, q).

Example:- We have a query ASKVARS (knows (John, x)), whom does John know? Answer can be found by finding all sentences in the knowledge base that unify with knows (John, x). Here are results of unification with four different sentences that might be in the knowledge base:

UNIFY (knows (John, x), knows (John, Jane)) = {x/Jane}

UNIFY (knows (John, x), knows (y, Bill)) = {x/Bill, y/John}

UNIFY (knows (John, x), knows (y, Mother (y))) = {y/John, x/Mother (John)}

UNIFY (knows (John, x), knows (x, Elizabeth)) = fail

The last unification fails because x cannot take on values John and Elizabeth at the same time. We can resolve the error by using two different variable or a different version of x as x_{17}. standardizing apart one of two sentences being unified.

UNIFY (knows (John, x), knows (x_{17}, Elizabeth)) = {x/Elizabeth, x_{17}/John}.

10) Describe Backward-chaining algorithm for first order definite clause.

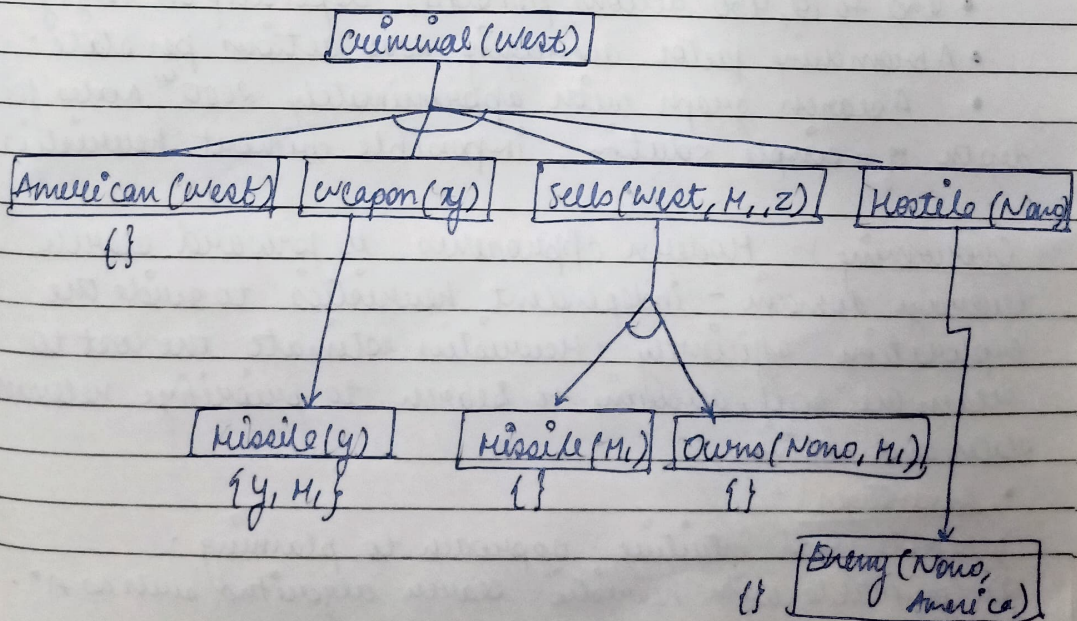
→ Backward-chaining is also known as a backward deduction or backward reasoning method when using an inference engine. A backward chaining algorithm is a form of reasoning, which starts with the goal and works backward, chaining through rules to find known facts that support the goal.

It is known as a top-down approach. Backward chaining is based on modus ponens inference rule. In backward chaining, the goal is broken into sub-goal or sub-goals to prove the facts true. It is a goal-driven approach, as a list of goals decides which rules are selected and used. The backward-chaining method mostly used a depth-first search strategy for proof.

function FOL-BC-ASK (KB, query) returns a generator of substitutions
    return FOL-BC-OR (KB, query, {})

generator FOL-BC-OR(KB, goal, θ) yields a substitution
    for each rule (lhs ⇒ rhs) in FETCH-RULES-FOR-GOAL (KB, goal) do
        (lhs, rhs) ← STANDARDIZE-VARIABLES ((lhs, rhs))
        for each θ' in FOL-BC-AND(KB, lhs, UNIFY (rhs, goal, θ)) do
            yield θ'

generator FOL-BC-AND(KB, goal, θ) yields a substitution
    if θ = failure then return
    else if LENGTH (goals) = 0 then yield θ
    else do
        first, rest ← FIRST (goals), REST (goals) do
        for each θ' in FOL-BC-OR (KB, SUBST (θ, first), θ)
            for each θ'' in FOL-BC-AND(KB, rest, θ') do
                yield θ''



13) Describe Forward (Progression) State-Space Search Algorithm with an example.

→ Forward (progression) state-space search is a planning approach where the search begins from the initial state and applies available actions to progress toward a goal state. Each state represents a snapshot of the environment, and transitions between states are dictated by the actions.

The search involves:-

1) Starting from the initial state.
2) Applying actions that result in new states.
3) Continuing this process until a state in the goal set is reached.

Challenges:- Despite its intuitive approach, forward state-space search faced criticism for being inefficient, particularly before 1998, due to reasons:

1) Exploration of Irrelevant Actions:- Forward search tends to explore many actions that have no relevance to the goal. Ex:- For the Task of buying a specific book using the action schema Buy(ISBN) with eight own (ISBN) the planner must consider upto 10 billion possible ISBNs to find the correct one. Uninformed exploration wastes computational resources.

2) Large State Space:- Many planning problems involve enormous state spaces, often with high branching factors, making exhaustive search computationally impossible.

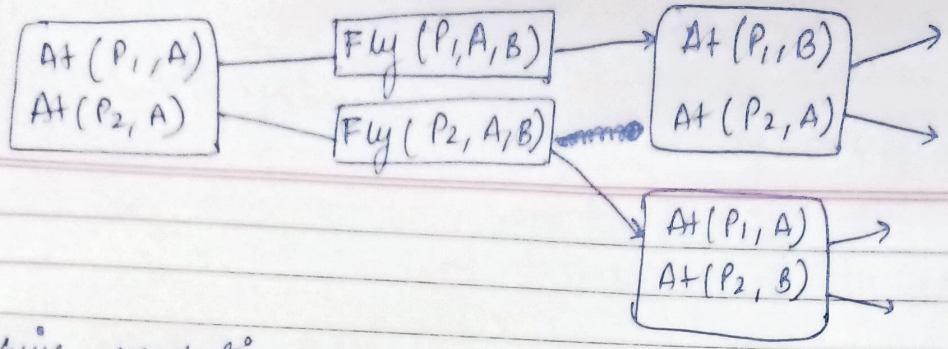
Example:- In an air cargo problem with 10 airports, 5 planes per airport, and 20 pieces of cargo, there are:

- 450 to 10,450 actions per state, depending on config.
- A branching factor averaging 2000 actions per state.
- A search graph with approximately 2000^{41} nodes for depth of a simple solution, impossible without heuristics.

Overcoming:- Modern approaches to forward search leverage domain-independent heuristics to guide the exploration efficiently. Heuristics estimate the cost to reach the goal, allowing to search to prioritize relevant states and actions.

- Advantages:-

1) Simple and intuitive approach to planning.
2) Compatible with heuristic search algorithms such as A*.



15) Write resolution proof that curiosity killed the cat?

→ First we express the original sentences, some background knowledge and the negated goal G in first-order logic:

A. $\forall x\ [\forall y\ Animal(y) \Rightarrow loves(x,y)] \Rightarrow [\exists y\ Loves(y,x)]$

B. $\forall x\ [\exists z\ Animal(z) \wedge kills(x,z)] \Rightarrow [\forall y\ \neg Loves(y,x)]$

C. $\forall x\ Animal(x) \Rightarrow Loves(Jack, x)$

D. $Kills(Jack, Tuna) \vee Kills(Curiosity, Tuna)$

E. $Cat(Tuna)$

F. $\forall x\ Cat(x) \Rightarrow Animal(x)$

$\neg G$. $\neg Kills(Curiosity, Tuna)$

Now, we apply conversion :-

A1. $Animal(F(x)) \vee Loves(G(x), x)$

A2. $\neg Loves(x, F(x)) \vee Loves(G(x), x)$

B. $\neg Loves(y,x) \vee \neg Animal(z) \vee \neg kills(x,z)$

C. $\neg Animal(x) \vee Loves(Jack, x)$

D. $Kills(Jack, Tuna) \vee Kills(Curiosity, Tuna)$

E. $Cat(Tuna)$

F. $\neg Cat(x) \vee Animal(x)$

$\neg G$. $\neg Kills(Curiosity, Tuna)$