1) a. This is False. Consider the case where $\beta \equiv A$, $\gamma \equiv \neg A$. Then $\beta \vee \gamma$ is valid. Therefore, we cannot say if $\alpha \models (\beta \vee \gamma)$ then $\alpha \models \beta$ or $\alpha \models \gamma$.

b. True. We can apply monotonicity to prove this. According to monotonicity, for any sentence γ and β, if $\alpha \models \gamma$, $\alpha \wedge \beta \models \gamma$. Therefore, if $\alpha \models \gamma$ or $\beta \models \gamma$ (or both) then $(\alpha \wedge \beta) \models \gamma$ is True.

NAME: ARNAB DEY
UMN ID: 5563169
EMAIL : DEY00011@UMN.EDU

c. True. This can be proved by the following truth table –

β	γ	$\beta \wedge \gamma$
F	F	F
F	T	F
T	F	F
T	T	T

Therefore, if $(\beta \wedge \gamma)$ is true in every model of α, then β and γ are true in every model of α, i.e. $\alpha \models \beta$ and $\alpha \models \gamma$.

2.a) 2.b) Please see attached code.

2)c) I got the following result from a single run of (2.b) –

m	n	m/n ratio	Prob. of satisfiability
30	10	3.00	1
30	15	2.00	1
30	20	1.50	1
40	10	4.00	0.78
40	15	2.67	1
40	20	2.00	1
50	10	5.00	0.49
50	15	3.33	0.99
50	20	2.50	1
60	10	6.00	0.18
60	15	4.00	0.85
60	20	3.00	1
70	10	7.00	0.01
70	15	4.67	0.55
70	20	3.50	0.99

observations :-

(1) For a fixed number of literals (n), if number of clauses (m) increases, probability of satisfiability decreases.

(2) If m/n ratio is ≤ 3.00, probability of satisfiability is almost 1.0.

(3) As m/n ratio increases probability of satisfiability decreases.

(4) If m/n ratio is > 6.00, probability of satisfiability is almost 0.

Analysis :-

For a clause with 3 literals, probability that the clause is true (assuming CNF form) $= \left(1 - \frac{1}{2^3}\right)$

Therefore, m independent clauses are satisfied with probability $\left(1 - \frac{1}{2^3}\right)^m$.

Therefore, as m increases, probability of satisfiability decreases.

Now, there are 2^n possible assignments of n literals. Therefore expected number of satisfiable conjunctions are $2^n \left(1 - \frac{1}{2^3}\right)^m$.

∴ If n increases, i.e. m/n ratio decreases, probability of satisfiability increases.

3.a) Given axioms →

$0 \leq 2$ ---------- (1)

$4 \leq 8$ ---------- (2)

$\forall x,\ x \leq x$ ---------- (3)

$\forall x,\ x \leq x+0$ ---------- (4)

$\forall x, y\ \ x+y \leq y+x$ ---------- (5)

$\forall w, x, y, z,\ w \leq y \wedge x \leq z \Rightarrow w+x \leq y+z$ ---------- (6)

$\forall x, y, z,\ x \leq y \wedge y \leq z \Rightarrow x \leq z$ ---------- (7)

We have to prove $4 \leq 2+8$.

Steps :-

(A) From (1), (2) and (6), applying $\{w/0, y/2, x/4, z/8\}$ infer

$0+4 \leq 2+8$ ---------- (8)

(B) From (8), (5), and (7), and using renamed variables $x5, y5$ for (5) and $x7, y7, z7$ for (7), we apply

$\{x5/4, y5/0, x7/4+0, y7/0+4, z/2+8\}$

infer,

$4+0 \leq 2+8$ ..... (9)

(C) From (4), (9), and (7), and using renamed variables $x4$ for (4) and $x71, y71, z71$ for (7), we apply

$\{x4/4, x71/4, y71/4+0, z71/2+8\}$

infer

$4 \leq 2+8$ ..... (10) [Proved]

3.b) we have to show backward-chaining proof of $4 \leq 2+8$.

[variable notation = xAB ≡ variable 'x' in equation (A) occurring 'B'th time]

	Resolutions
Goal G0: $4 \leq 2+8$	Resolve using (7), apply $\{x71/4, z71/2+8\}$
Goal G1: $4 \leq y71$	Resolve using (4), apply $\{x41/4, y71/4+0\}$. Result = Success.
Goal G2: $4+0 \leq 2+8$	Resolve using (7), apply $\{x72/4+0, z72/2+8\}$.
Goal G3: $4+0 \leq y72$	Resolve using (5), apply, $\{x51/4, y51/0, y72/0+4\}$ Result = Success
Goal G4: $0+4 \leq 2+8$	Resolve using (6), apply $\{w61/0, x61/4, y61/2, z61/8\}$.

Goal G5: $0 \leq 2$	Resolve using (1) Result = Success
Goal G6: $4 \leq 8$	Resolve using (2). Result = Success
$\therefore$ G4 succeeds	
$\therefore$ G2 succeeds.	
$\therefore$ G0 succeeds.	

[Proved]

4) Predicates :

① Door(x) := x is a door.

② Open(x) := x is open.

③ Bowl(x) := x is a bowl.

④ Marble(x) := x is a marble.

⑤ In(x, y) := x is in y.

⑥ Hidden(x) := x is hidden.

a) Initial State :-

Init(In(Marble1, Bowl1) ∧ In(Marble2, Bowl4)
∧ Hidden(Bowl4) ∧ Hidden(Marble2) ∧
Bowl(Bowl1) ∧ Bowl(Bowl2) ∧ Bowl(Bowl3)
∧ Door(ExitDoor) ∧ Door(ClosetDoor) ∧
Marble(Marble1) ∧ Marble(Marble2) ∧
¬ Open(ExitDoor) ∧ ¬ Open(closetDoor))

4.b) Goal State :-

Goal (Open (ExitDoor) ∧ (∃ x Marble(x) ∧ In(x, Bowl1)) ∧ (∃ y Marble(y) ∧ In(y, Bowl2)) ∧ (x ≠ y))

4.c) PDDL action schema :-

Action (Put (m, t, f),

PRECOND : Marble (m) ∧ Bowl (t) ∧ Bowl(f) ∧ (t≠f) ∧ In (m, f) ∧ ¬ Hidden (m) ∧ ¬ Hidden (t) ∧ ¬ Hidden (f),

EFFECT : In(m, t) ∧ ¬ In (m, f)) ∧ when (∃ x Marble (x) ∧ In (x, Bowl3)): open (closetDoor) ∧ ¬ Hidden (Bowl4) ∧

$\ulcorner$ Hidden (Marble 2) $\wedge$

when $\ulcorner$ In (Marble 1, Bowl 3) $\wedge$ $\ulcorner$ In (Marble 2, Bowl 3) : $\ulcorner$ Open (Closet Door) $\wedge$ Hidden (Bowl 4)

$\wedge$ when $(\exists x$ Marble $(x) \wedge$ In $(x$, Bowl 4)) : Hidden $(x) \wedge$

$\wedge$ when $\ulcorner$ In (marble 1, Bowl 1) $\wedge \ulcorner$ In (Marble 2, Bowl 1) : $\ulcorner$ Open (Exit Door) $\wedge$ when $\ulcorner$ In (Marble 1, Bowl 2) $\wedge \ulcorner$ In (Marble 2, Bowl 2) : $\ulcorner$ Open (Exit Door))

Action (Open Exit Door (x, y),

PRECOND: Marble $(x) \wedge$ Marble $(y) \wedge (x \neq y)$,

EFFECT: when In $(x$, Bowl 1) $\wedge$ In $(y$, Bowl 2): Open (Exit Door) $\wedge$ when In $(y$, Bowl 1) $\wedge$ In $(x$, Bowl 2) : Open (Exit Door)) .

These are the required PDDL action schema.

A plan to achieve the goal :-

Put (Marble1, Bowl3, Bowl1)
Put (Marble2, Bowl2, Bowl4)
Put (Marble1, Bowl1, Bowl3)
OpenExitDoor (Marble1, Marble2)

5) a) Knight's tour! uninformed search :-

I would try to use depth first search using backtracking. Nodes of the graphs would be the position of the knight on the board (0-63). For each position (node), there are 8 possible next position as shown in Fig.1

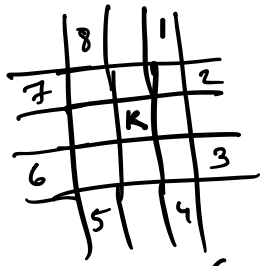

Fig 1: K: Knight, (1-8): Next possible moves

DFS would choose one of them

and append that node to a list to track what all nodes are visited and in what sequence. The goal state would be to have the list contain 0 - 63 without repeatation. If it encounters a node which is already there in the visited list, my DFS would back track to the previous node & try other children of that node. At the end, we would get a sequence of nodes, starting from the initial position, to complete a path with visiting all positions only once.

Knight's tour : Planning :-

I would use the following predicates

① At(x, y) ≡ Knight's position at row x, column y.

② visited(x, y) ≡ True if (x, y) coordinate is visited by the knight.

③ DiffByTwo(f, t) ≡ True if row/column from where (f) the knight wants to move to the next row/column (t) index differs by 2.

④ DiffByOne(f, t) = True if row/column from where (f) the knight wants to move to the next row/column (t) index differs by 1.

⑤ valid(x, y) ≡ if (x, y) is a valid position on the board.

Goal state would be conjunction of visited(x,y) predicates for all x, y $\in \{0, 1, \ldots, 63\}$.

Init state would be At(x,y) where x,y would be the initial position.

Actions would be 'Move2Col1Row' and 'Move1Col2Row'. They will have preconditions to check if the next location is valid. Effect would be next location is visited.

Analysis :- As there are 8 possible moves from any given position, and the depth of the tree is 8^2 (as each node

corresponds to one position on the 8x8 board), the time complexity of DFS would be $O(8^{8^2})$. And if I use backtracking, the space complexity would be $O(8^2)$. The time complexity for uninformed search is large. On the other hand, for planning, we need to define 8^2 objects and for the goal state, we have to write 8^2 predicates in a conjunctive form. Also we have to write many ground predicates to indicate which rows/columns differ by 2 or 1. Thus the space complexity would be very high.

We would not gain much in time complexity also as all the logical relations between the states are completely characterized in DFS also. Thus I would choose backtracking DFS for Knight's tour problem.

5.b) Crossword puzzle : Local search :-

Starting from a valid initial position (i.e. starting of a horizontal available grid position whose left position is either a black square or puzzle boundary), my search algorithm would

choose words of length fitting into the horizontal position until a black square (from a dictionary, the words would be chosen). The successors of that word would be the words that fit the vertical positions for each letter of that word. To place a word, we have to use 'Regex' to verify which of the many possible words, in the dictionary, fits by satisfying positions of all the letters that are already present. The heuristic could be based on relative frequency of

english letters (so that it helps us in having more options to fill the rest of the puzzle in future). The goal state would be a puzzle with no square left empty.

Crossword puzzle: Constraint Satisfaction

A word would be a variable. The constraints on a variable (word) would be it has to be a string of certain length (based on the position on the puzzle) and it needs to have letters at the positions, wherever it intersects another word, same as the

intersecting word. Another variable is required which would keep track of all other intersecting words and corresponding intersection positions. The domain of each variable would be a dictionary. Then we can use backtracking with least constraining value heuristic to have more options to fill out in future.

Analysis :- I believe, for crossword puzzle, constraint satisfaction would perform better. CSP would require less memory because as the words are placed, the domains of other variables (words

in other positions on the crossword) shrinks significantly. Because of this the runtime of csp would be comparatively faster than local search. In local search it is very difficult to handle the dynamically changing tree (as in the case of crossword puzzle). Thus a naive choice would be to check all successors of a word (basically whole dictionary) to check which successor meets the crossword rules. This would increase the runtime significantly.

5.c) Minesweeper: Knowledge based agent

Each cell is represented as a variable that can take two values, 0 or 1, since either the cell is not having a mine or there is a mine. Now, consider the following position -

Minesweeper

Then we can write $x_1 + x_2 + x_3 + x_4 + x_5 = 4$.

Now, as x_1, x_2, x_3, x_4, x_5 can have values 0/1, we can enumerate all possible assignments & convert them to CNF form, then add them

to our knowledge base. The agent would flag cells for which truth value is not known & reveal those cell for which the inferred value is 0 (no mine). Upon revealing a cell, the knowledge base is updated as explained above. If we do not assume that total number of mines is known, then the agent would continue until all cell values are inferred.

Minesweeper: Goal based agent:

The predicates would be -

(1) Safe(x) ≡ True if a cell x is not having mine.

(2) Flag(x) ≡ True if a cell x is flagged.

(3) IsCovered(x) ≡ True if cell x is uncovered.

(4) Adjacent(x,y) ≡ True if x,y are adjacent.

Actions could be uncover(x) for which the precondition is ¬IsCovered(x) ∧ Safe(x). The effect would be to reveal the number shown in that cell and update the values of all variables that are adjacent to it (by inferring).

Goal state is the conjunction of safe(x) for all x.

Analysis :- I believe for minesweeper, knowledge-based approach would be better. Because unless a cell is uncovered and a number is shown, an agent cannot look ahead and infer states of cells which are not adjacent to any uncovered cell. The environment is not fully observable in minesweeper. Therefore, having a final goal in such an environment where states cannot be

observed unless actions are taken, having goal based agent would not help. Runtime may be comparable for both knowledge-based & goal based. However, problem structure wise knowledge based approach would be easier to implement.