Question 1

Predicates:

carmichael(X): Person X belongs to Carmichael Gym.
runner(X): Person X is a runner.
footballer(X): Person X is a footballer.
like(X, Y): Person X likes Y.

FOPL:

a. John, Pitt, and Lisa are members of Carmichael gym.
   carmichael(John) ∧ carmichael(Pitt) ∧ carmichael(Lisa)

b. Each member of this gym is either runner, a footballer, or both.
   ∀X (carmichael(X) ⇒ (runner(X) ∨ footballer(X)))

c. Neither of the footballers like rain.
   ∀X (footballer(X) ⇒ ¬like(X, rain))

d. All the runners like sunny weather.
   ∀X (runner(X) ⇒ like(X, sunny))

e. Pitt hates whatever John likes, and likes whatever John hates.
   ∀X ((like(John, X) ⇒ ¬like(Pitt, X)) ∧ (¬like(John, X) ⇒ like(Pitt, X)))

f. John likes rain and sunny weather.
   like(John, rain) ∧ like(John, sunny)

CNF:

1. carmichael(John)
2. carmichael(Pitt)
3. carmichael(Lisa)
4. ¬carmichael(X1) ∨ runner(X1) ∨ footballer(X1)
5. ¬footballer(X2) ∨ ¬like(X2, rain))
6. ¬runner(X3) ∨ like(X3, sunny))
7. ¬like(John, X4) ∨ ¬like(Pitt, X4))
8. like(John, X5) ∨ like(Pitt, X5))
9. like(John, rain)
10. like(John, sunny)

Conclusion: There is a member of Carmichael gym that is a footballer, but not a runner.
∃X (carmichael(X) ∧ footballer(X) ∧ ¬runner(X))

Negated conclusion: ¬carmichael(X) ∨ ¬footballer(X) ∨ runner(X) … (De Morgan's Law)

1. carmichael(John)
2. carmichael(Pitt)
3. carmichael(Lisa)
4. ¬carmichael(X1) ∨ runner(X1) ∨ footballer(X1)
5. ¬footballer(X2) ∨ ¬like(X2, rain))
6. ¬runner(X3) ∨ like(X3, sunny))
7. ¬like(John, X4) ∨ ¬like(Pitt, X4))
8. like(John, X5) ∨ like(Pitt, X5))
9. like(John, rain)
10. like(John, sunny)

--	
11. ¬carmichael(X6) ∨ ¬footballer(X6) ∨ runner(X6)	Negated conclusion
--	
12. ¬footballer(John) ∨ runner(John)	11 + 1 {John/X6}
13. ¬carmichael(John) ∨ runner(John)	12 + 4 {John/X1}
14. ¬carmichael(John) ∨ like(John, sunny)	13 + 6 {John/X3}
15. ¬carmichael(John) ∨ ¬like(Pitt, sunny)	14 + 7 {sunny/X4}
16. ¬carmichael(John) ∨ like(John, sunny)	15 + 8 {sunny/X5}
DEAD END	
17. ¬footballer(Pitt) ∨ runner(Pitt)	11 + 2 {Pitt/X6}
18. ¬carmichael(Pitt) ∨ runner(Pitt)	17 + 4 {Pitt/X1}
19. ¬carmichael(Pitt) ∨ like(Pitt, sunny)	18 + 6 {Pitt/X3}
20. ¬carmichael(Pitt) ∨ ¬like(John, sunny)	19 + 7 {sunny/X4}
21. ¬carmichael(Pitt) ∨ like(Pitt, sunny)	20 + 8 {sunny/X5}
DEAD END	
22. ¬carmichael(Pitt)	20 + 10 { }
DEAD END	
23. ¬footballer(Lisa) ∨ runner(Lisa)	11 + 3 {Lisa/X6}
24. ¬carmichael(Lisa) ∨ runner(Lisa)	24 + 4 {Lisa/X1}
25. ¬carmichael(Lisa) ∨ like(Lisa, sunny)	18 + 6 {Lisa/X3}
DEAD END	
26. ¬carmichael(X1) ∨ runner(X1)	11 + 4 {X1/X6}
27. ¬carmichael(X3) ∨ like(X3, sunny))	26 + 6 {X3/X1}
28. ¬carmichael(Pitt) ∨ ¬like(John, sunny)	27 + 7 {Pitt/X3, sunny/X4}
29. ¬carmichael(Pitt)	28 + 10 { }
30. □	29 + 2 { }

It is proved that the contradiction of the conclusion is an empty clause (false) and the conclusion is, thus, derivable.

(Pitt is the member of Carmichael gym who is a footballer and not a runner)

<u>**Question 2**</u>

Constants: 1, 2, 3, empty

Predicates:

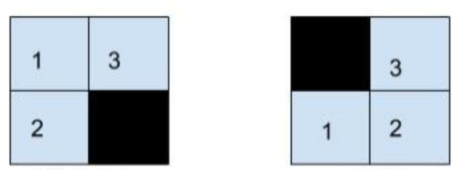
At(t, r, c) : Tile t is at row r and column c. Top-most row is 1. Left-most column is 1.
Pos(n) : n is a position index (1 or 2)
Tile(t) : t is a tile

Action Schemas:

action(Up(t, c),
    PRECOND: At(t, 2, c) ∧ At(empty,1,c) ∧ Pos(c) ∧ Tile(t)
    EFFECT: ¬At(t, 2, c) ∧ At(t,1,c) ∧ ¬At(empty, 1, c) ∧ At(empty, 2, c)
action(Down(t, c),
    PRECOND: At(t, 1, c) ∧ At(empty, 2, c) ∧ Pos(c) ∧ Tile(t)
    EFFECT: ¬At(t, 1, c) ∧ At(t, 2, c) ∧ ¬At(empty, 2, c) ∧ At(empty, 1, c)
action(Right(t, r),
    PRECOND: At(t, r, 1) ∧ At(empty, r, 2) ∧ Pos(r) ∧ Tile(t)
    EFFECT: ¬At(t, r, 1) ∧ At(t, r, 2) ∧ ¬At(empty, r, 2) ∧ At(empty, r, 1)
action(Left(t, r),
    PRECOND: At(t, r, 2) ∧ At(empty, r, 1) ∧ Pos(r) ∧ Tile(t)
    EFFECT: ¬At(t, r, 2) ∧ At(t, r, 1) ∧ ¬At(empty, r, 1) ∧ At(empty, r, 2)

1. Draw the plan graph of this problem, starting from the initial state described below, and continue until you first reach the goal state.
   Init(At(1, 1, 1) ∧ At(2, 2, 1) ∧ At(3, 1, 2) ∧ At(empty, 2, 2) ∧ Pos(1) ∧ Pos(2) ∧ Tile(1) ∧ Tile(2) ∧ Tile(3))
   Then show the solution plan. Don't forget, you will need a negated literal for every ground literal whose predicate symbol and arguments are in the lexicon but not in the initial state description. And don't forget the continuation actions.

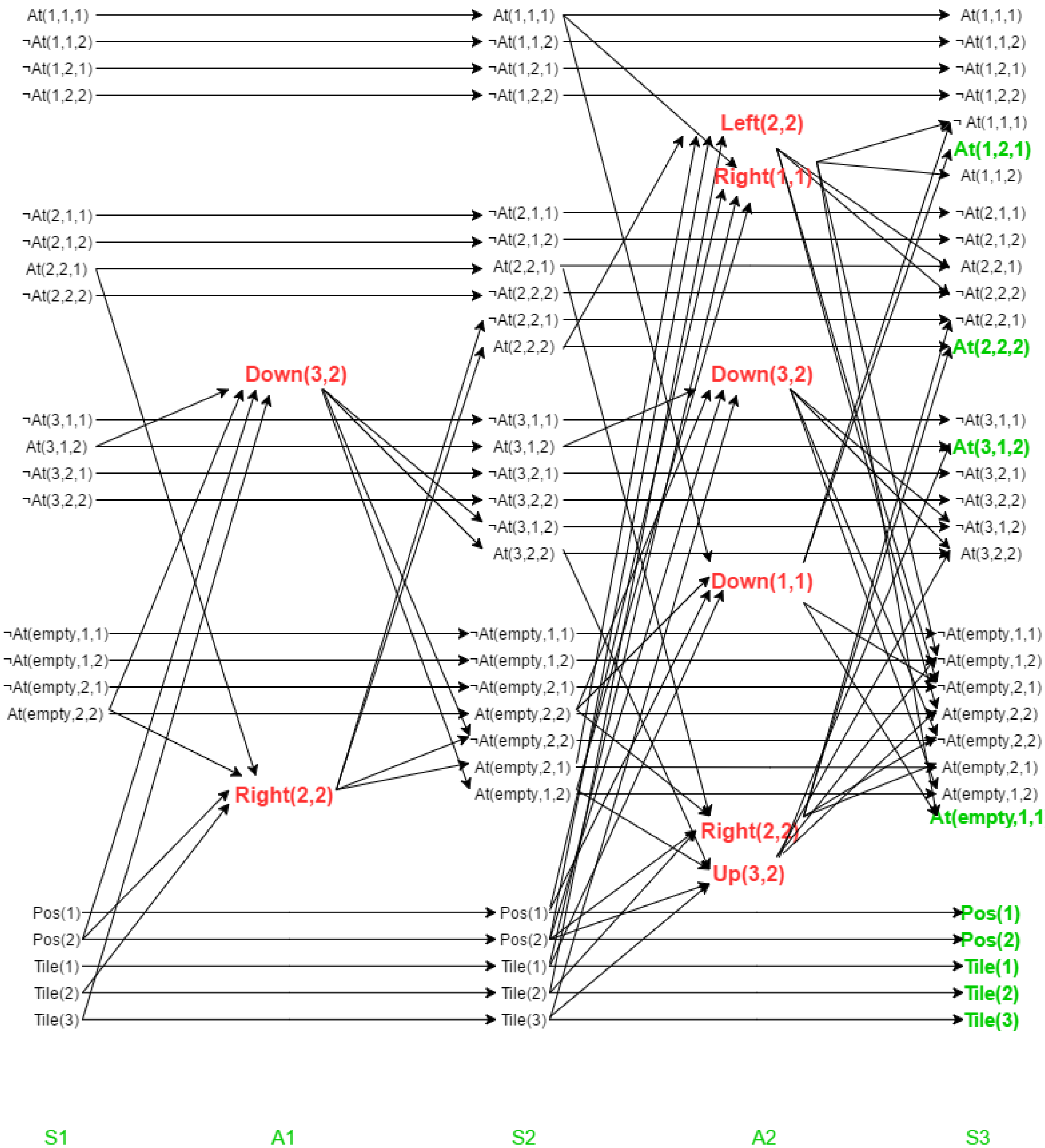
   Ans:



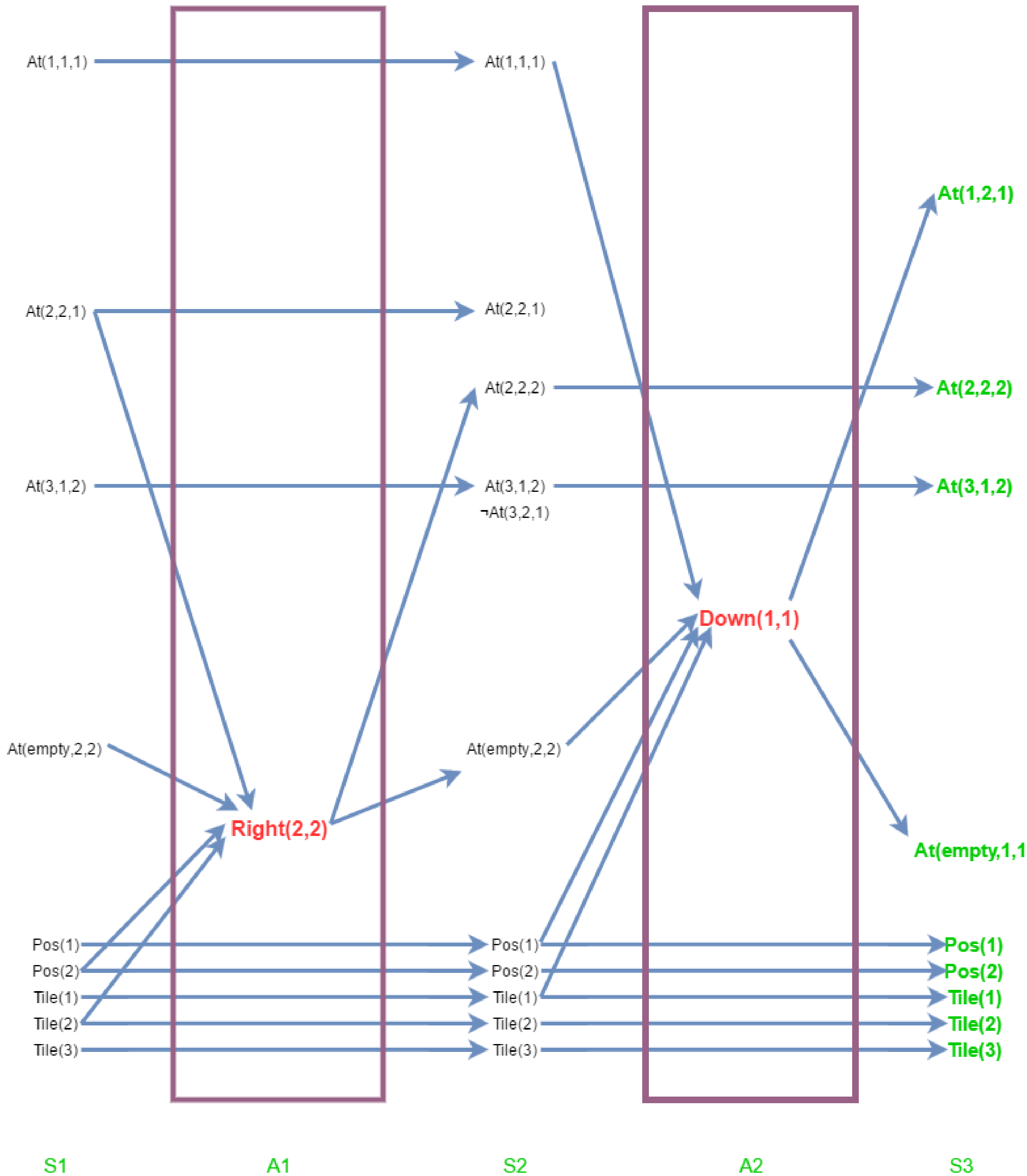
Initial state   goal state

The goal state can be represented as follows:
Goal(At(1, 2, 1) ∧ At(2, 2, 2) ∧ At(3, 1, 2) ∧ At(empty, 1, 1) ∧ Pos(1) ∧ Pos(2) ∧ Tile(1) ∧ Tile(2) ∧ Tile(3))

The planning graph showing initial state, paths leading to all the possible states and the goal state is drawn below: (lexicons in the goal state description are shown in green)

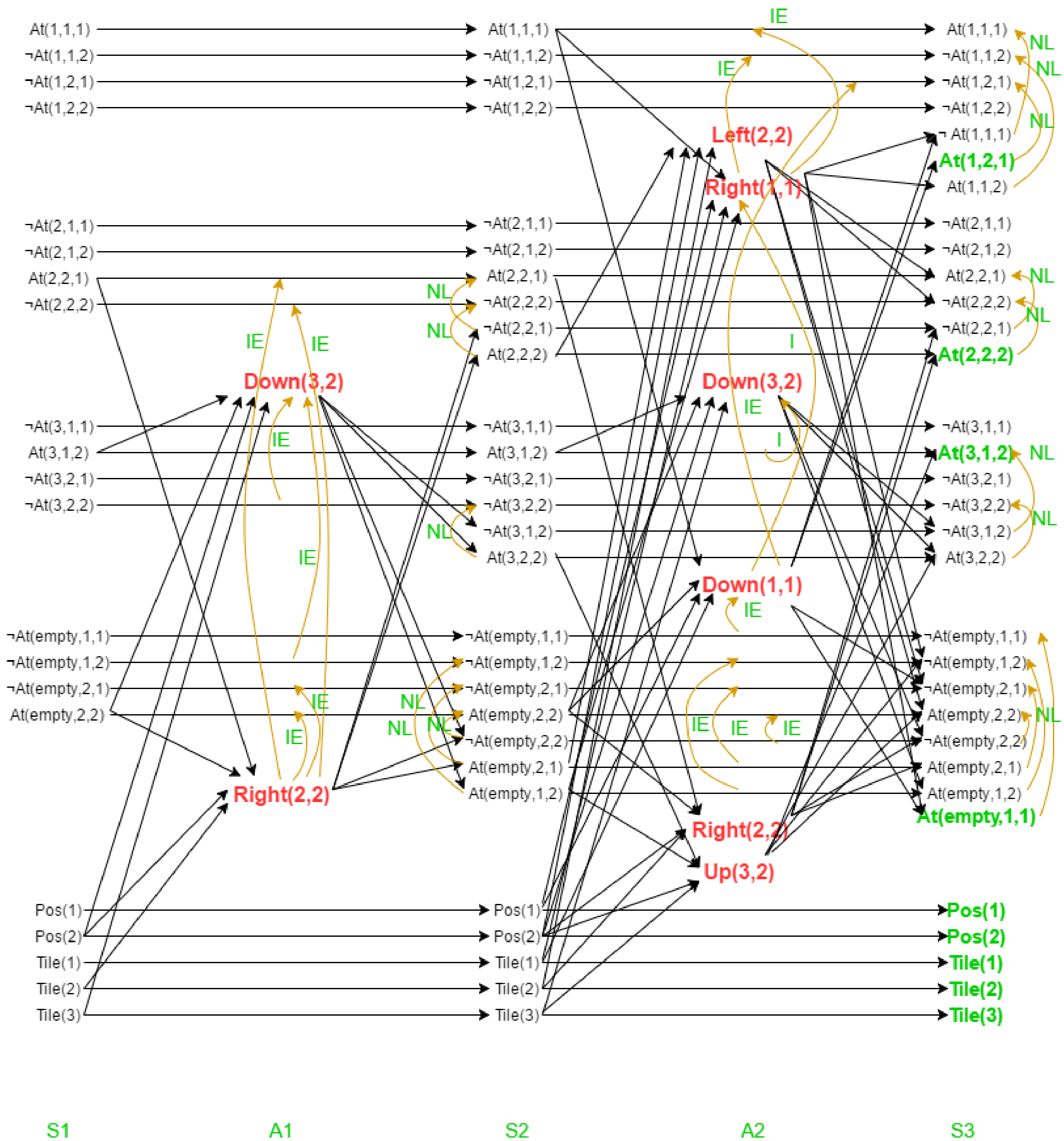


The solution path extraction is shown below. AND-OR backward search is used to extract the solution path.
OR: It considers one of the possible effects of a particular action leading to the next state.
AND: It considers all the preconditions required for a particular action to take place.



The solution path is: [right(2, 2), down(1,1)]

2. Identify, with curved vertical edges, the mutexes in this graph. Label the mutexes with their type (NL, IE, I, CN, IS). Look for them in that heuristic order. For each pair, one type of mutex is enough; things are either mutex or not, and we don't need multiple reasons. Some mutex types may not occur at all.



S_i - State Layer i
A_i - Action Layer i
NL - Negated Literals
IE - Inconsistent Effects
I - Interference

Heuristic order for mutex is (NL, IE, I, CN, IS).

- Negated Literals (NL): I_i and I_j are NLs if they are complementary to each other.
e.g. At(2, 2, 1) and ¬At(2, 2, 1) are mutexes by NL. Because, they are complements of each other. Similarly, all the NLs are shown in the above figure.

- Inconsistent effects (IE): Actions A_i and A_j are mutexes by IE, if the effect of A_i is NL of the effect of A_j.
e.g. Right(2, 2) and continuation action At(2, 2, 1) are mutexes by IE. Because, the effect of Right(2, 2) i.e. ¬At(2, 2, 1) and effect of the continuation action At(2, 2, 1) i.e. At(2, 2, 1) are NLs. Similarly, all the IEs are shown in the above figure.

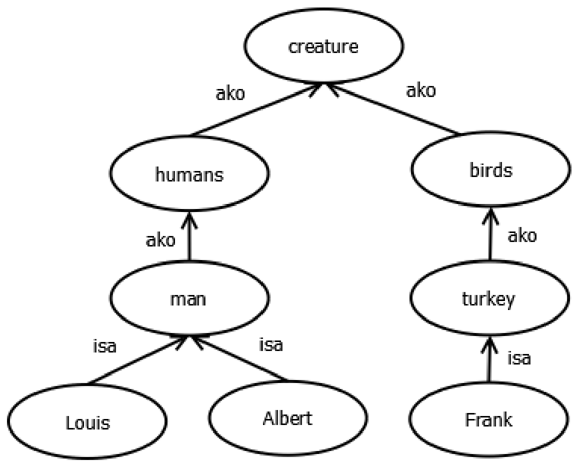
- Interference (I): If the effect of action A_i is the NL of a precondition for action A_j, then they are mutexes by I.
e.g. Right(1, 1) and Down(1, 1) are mutexes by I. Because, the precondition for Down(1, 1) i.e. At(1, 1, 1) is NL of the effect of Right(1, 1) i.e. ¬At(1, 1, 1). Similarly, all the Is are shown in the above figure.

Question 3

Facts and Rules:

Creatures come in two types: humans and birds. One type of human is a man. One type of bird is a turkey. Louis is a man. Albert is a man. Frank is a turkey.

1. Draw this taxonomy as a graph, with ``creature'' at the root, and label the edges with AKO or ISA, whichever is appropriate.



2. Suppose these facts were represented by seven FOPL facts of the form edge(<sourceNode>, <linkType>, <destinationNode>). Implement these facts as Prolog facts. Using as a top level rule head the syntax rel(SourceNode, RelationshipType, DestinationNode) and any other predicates you need, write a set of one or more rules to allow the inference that:
    1. Louis is a man, Louis is a human, and Louis is a creature.
    2. Albert is a man, Albert is a human, and Albert is a creature.
    3. Frank is a turkey, Frank is a bird, and Frank is a creature.

   Your rules should follow strict Prolog syntax, and should allow inference over hierarchies of **any** depth, not just the depth in this example.

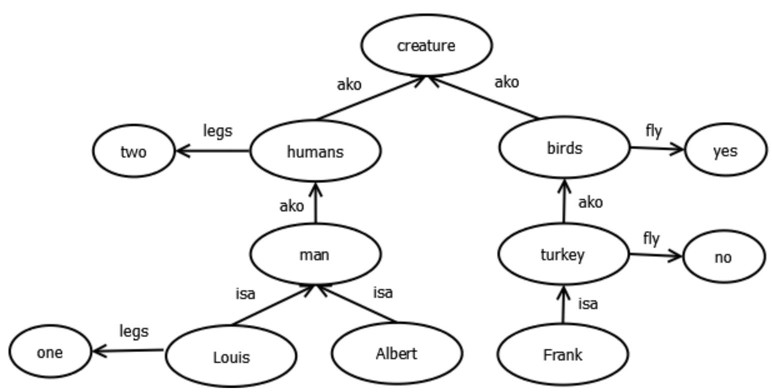
   Ans:
   Edges can be added by implementing following facts in PROLOG:
   1. edge(human, ako, creature).
   2. edge(bird, ako, creature).
   3. edge(man, ako, human).
   4. edge(turkey, ako, bird).
   5. edge(louis, isa, man).
   6. edge(albert, isa, man).
   7. edge(frank, isa, turkey).

For the inference mentioned in the question, detailed queries are written in the readme file.

Base case: rel(A, Slot, B):- edge(A, Slot, B).
Recursive case: rel(A, Slot, B):- edge(A, Slot, Z), rel(Z, isa, B).
rel(A, Slot, B):- edge(A, Slot, Z), rel(Z, ako, B).

Here, recursive call to rel( ) will find out all the possible combinations of node Louis with its parent nodes which have direct or indirect transitivity. (e.g. if Louis ISA man and man AKO human, then Louis ISA human)
Recursion generalizes this transitivity over any depth. There is no restriction for the path length involved in this transitivity.

3. Now add nodes and edges to the network to represent the knowledge that humans normally have two legs and birds can normally fly, but Louis has one leg and turkeys cannot fly. Using fact syntax such as property(<node>, legs, two) and property(<node>, fly, no), indicate which new facts will be necessary, and show in your network sketch from Part (a) where they should be added.
Ans: The following new facts should be added:
   1. Humans have two legs: property(human, legs, two).
   2. Louis has one leg: property(louis, legs, one).
   3. Birds can fly: property(bird, fly, yes).
   4. Turkeys don't fly: property(turkey, fly, no).



4. Add rule(s) to allow inference that (i) Frank cannot fly, (ii) Albert has two legs. and (iii) Louis has one leg. Your new rules will need to use the new facts from Part (c).
Ans:
```
hasProperty(A, Prop, Value):-  property(A, Prop, Value).
hasProperty(A, Prop, Value):-  edge(A, isa, Z),
                               hasProperty(Z, Prop, Value),
                               \+ property(A, Prop, _).
hasProperty(A, Prop, Value):-  edge(A, ako, Z),
                               hasProperty(Z, Prop, Value),
                               \+ property(A, Prop, _).
```

This will help in overriding the default attribute of the ancestor.
e.g. All humans have two legs, but Louis has only one leg. Here, the default attribute for humans is two legs, but this can be overrode using the above rules.

Rule 1 checks whether A has the local attribute for the property or not? e.g. Louis has one leg, man has two legs, birds can fly, etc. This rule works as the base case for the recursive cases which are rule 2 and rule 3.

Rule 2 checks whether A's ancestor has any default attribute or not. If A has any local attribute, then this will override the attribute inherited by its ancestor. e.g. Louis' ancestor is man. The default value for legs is two for man. Louis is expected to have the same value for legs as Louis ISA man. But, Louis himself has a local attribute i.e. one leg. So, this rule will override the legs value in the property for Louis and change it to one instead of default two.

Rule 3 is similar to rule 2, it just works for AKO relations with the ancestors. i.e. set-set relationship in the hierarchy, while rule 2 works for ISA relations i.e. member-set relationship.

For the above mentioned examples:

(i) Frank cannot fly
Here, Frank ISA turkey and turkey AKO bird.
So, turkey will override the default value for fly attribute as NO (replacing YES, which is inherited from bird). Then, as Frank doesn't have any local attribute value, the NO will remain as default for Frank.

(ii) Albert has two legs
Here, Albert ISA man and man AKO humans.
So, man will inherit value two for legs from humans. As, Albert doesn't have any local attribute value, the two will remain as default for Albert.

(iii) Louis has one leg
Here, Louis ISA man and man AKO humans.
So, man will inherit value two for legs from humans. Then, as Louis has a local attribute of one leg, this will override the default value two for him.

As there are different implementations possible for the same function, the way I implemented the queries are like mentioned above.
PROLOG QUERIES FOR ALL THESE QUESTIONS ARE PROVIDED IN README FILE