CS 371/671 Mid-term Study Guide

To be discussed in class on October 2nd

1 Agents

Question 2.2 Let us example the rationality of various vacuum cleaner agent functions.

a. Show that the simple vaccum-cleaner agent function described in Figure 2.3 is indeed rational under the assumptions in page 38.

Ans: It suffices to show that for all possible actual environments (i.e., all dirt distributions and initial locations), this agent cleans the squares at least as fast as any other agent. This is trivially true when there is no dirt. When there is dirt in the initial location and none in the other location, the world is clean after one step; no agent can do better. When there is no dirt in the initial location but dirt in the other, the world is clean after two steps; no agent can do better. When there is dirt in both locations, the world is clean after three steps; no agent can do better. (Note: in general, the condition stated in the first sentence of this answer is much stricter than necessary for an agent to be rational.)

b. Describe a rational agent function for the case in which each movement costs one point. Does the corresponding agent program require an internal state?

Ans: The agent in (a) keeps moving backwards and forwards even after the world is clean. It is better to do NoOp once the world is clean (the chapter says this). Now, since the agents percept doesnt say whether the other square is clean, it would seem that the agent must have some memory to say whether the other square has already been cleaned. To make this argument rigorous is more difficultfor example, could the agent arrange things so that it would only be in a clean left square when the right square was already clean? As a general strategy, an agent can use the environment itself as a form of external memorya common technique for humans who use things like appointment calendars and knots in handkerchiefs. In this particular case, however, that is not possible. Consider the reflex actions for [A,Clean] and [B,Clean]. If either of these is NoOp, then the agent will fail in the case where that is the initial percept but the other square is dirty; hence, neither can be NoOp and therefore the simple reflex agent is doomed to keep moving. In general, the problem with reflex agents is that they have to do the same thing in situations that look the same, even when the situations are actually quite different. In the vacuum world this is a big liability, because every interior square (except home) looks either like a square with dirt or a square without dirt.

c. Discuss possible agent designs for the cases in which clean squares can become dirty and the geography of the environment is unknown. Does it make sense to learn from its experience in these cases? If so, what should it learn? If not, why not?

Ans: If we consider asymptotically long lifetimes, then it is clear that learning a map (in some form) confers an advantage because it means that the agent can avoid bumping into walls. It can also learn where dirt is most likely to accumulate and can devise an optimal inspection strategy.

Question 2.3 For each of the following assertions, say whether it is true or false and support your answer with examples or counterexamples.

- An agent that senses only partial information about the state cannot be perfectly rational.

  Ans: False. Perfect rationality refers to the ability to make good decisions given the sensor information received.

- There exists task environments in which no pure reflex agent can behave rationally.

  Ans: True. A pure reflex agent ignores previous percepts, so cannot obtain an optimal state estimate in a partially observable environment. For example, correspondence chess is played by sending moves; if the other players move is the current percept, a reflex agent could not keep track of the board state and would have to respond to, say, a4 in the same way regardless of the position in which it was played.

- There exists task environments in which every agent is rational.

  Ans: True. For example, in an environment with a single state, such that all actions have the same reward, it doesnt matter which action is taken. More generally, any environment that is reward-invariant under permutation of the actions will satisfy this property.

- The input to an agent program is the same as the input to agent function.

  Ans: False. The agent function, notionally speaking, takes as input the entire percept sequence up to that point, whereas the agent program takes the current percept only.

- Every agent is implementable by some program/machine combination.

  Ans: False. For example, the environment may contain Turing machines and input tapes and the agents job is to solve the halting problem; there is an agent function that specifies the right answers, but no agent program can implement it. Another example would be an agent function that requires solving intractable problem instances of arbitrary size in constant time.

- Suppose an agent selects its actions uniformly at random from a set of possible actions, there exists a deterministic task environment for which this agent is optimal.

  Ans: True. This is a special case of (c); if it doesnt matter which action you take, selecting randomly is rational.

- It is possible for an agent to be perfectly rational in two distinct environments.

  Ans: True. For example, we can arbitrarily modify the parts of the environment that are unreachable by any optimal policy as long as they stay unreachable.

Question 2.13 Discuss possible agent programs for the following stochastic versions of the vacuum world.

a. Murphy's Law: 25% of the time, the *Suck* action fails to clean floor when dirty and deposits dirt if clean. How will your agent program be affected if the sensor gives the wrong answer 10% of the time?

   Ans: For a reflex agent, this presents no additional challenge, because the agent will continue to *Suck* as long as the current location remains dirty. For an agent that constructs a sequential plan, every *Suck* action would need to be replaced by Suck until clean. If the dirt sensor can be wrong on each step, then the agent might want to wait for a few steps to get a more reliable measurement before deciding whether to *Suck* or move on to a new square. Obviously, there is a trade-off because waiting too long means that dirt remains on the floor (incurring a penalty), but acting immediately risks either dirtying a clean square or ignoring a dirty square (if the sensor is wrong). A rational agent must also continue touring and checking the squares in case it missed one on a previous tour (because of bad sensor readings). it is not immediately obvious how the waiting time at each square should change with each new tour. These issues can be clarified by experimentation, which may suggest a general trend that can be verified mathematically. This problem is a partially observable Markov decision processsee Chapter 17. Such problems are hard in general, but some special cases may yield to careful analysis.

b. Small Children: At each time step, there is a 10% chance of each clean square becoming dirty. Can you come up with a rational agent in this case?

   Ans: In this case, the agent must keep touring the squares indefinitely. The probability that a square is dirty increases monotonically with the time since it was last cleaned, so the rational strategy is, roughly speaking, to repeatedly execute the shortest possible tour of all squares. (We say roughly speaking because there are complications caused by the fact that the shortest tour may visit some squares twice, depending on the geography.) This problem is also a partially observable Markov decision process.

2 Search and Heuristics

1. Imagine a car-like agent to exit a maze like the one shown below. The agent is directional and at all times faces some direction $d \in (N, S, E, W)$. With a single action, the agent can either move forward at an adjustable velocity v or turn. The turning actions are left and right, which change the agent's direction by 90 degrees. Turning is only permitted when the velocity is zero (and leaves it at zero). The moving actions are $fast$ and $slow$. $Fast$ increments the velocity by 1 and $slow$ decrements the velocity by 1; in both cases the agent then moves a number of squares equal to its NEW adjusted velocity. Any action which would collide with a wall crashes the agent and is illegal. Any action which would reduce v below 0 or above a maximum speed V_{max} is also illegal. The agent's goal is to

find a plan which parks it (stationary) on the exit square using as few actions (time steps) as possible.

As an example: if the agent shown were initially stationary, it might

first turn to the east using (right), then move one square east using fast, then two more squares east using fast again. The agent will of course have to slow to turn.

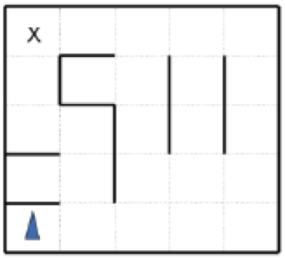

Figure 1:

a. If the grid is M by N, what is the size of the state space? Justify your answer. You should assume that all configurations are reachable from the start state.
Ans: The size of the state space is $4MN(V_{max}+1)$. The state representation is (direction facing; x; y; speed). Note that the speed can take any value in $\{0, ..., V_{max}\}$.

b. What is the maximum branching factor of this problem? You may assume that illegal actions are simply not returned by the successor function. Briefly justify your answer.
Ans: The maximum branching factor is 3, and this happens when the agent is stationary. While stationary it can take the following 3 actions - fast, left, right.

c. Is the Manhattan distance from the agent's location to the exit's location admissible? Why or why not?
Ans: No, Manhattan distance is not an admissible heuristic. The agent can move at an average speed of greater than 1 (by first speeding up to V_{max} and then slowing down to 0 as it reaches the goal), and so can reach the goal in less time steps than there are squares between it and the goal. A specific example: the target is 6 squares away, and the agent's velocity is already 4. By taking only 4 slow actions, it reaches the goal with a velocity of 0.

d. State and justify a non-trivial admissible heuristic for this problem which is not the Manhattan distance to the exit.
Ans: There are many answers to this question. Here are a few, in order of weakest to strongest:
(a) The number of turns required for the agent to face the goal.
(b) Consider a relaxation of the problem where there are no walls, the agent can turn and change speed arbitrarily. In this relaxed problem, the agent would move with V_{max}, and then suddenly stop at the goal, thus taking $d_{manhattan}/V_{max}$ time.
(c) We can improve the above relaxation by accounting for the the deceleration dynamics. In this case the agent will have to slow down to 0 when it is about to reach the goal. Note that this heuristic will always return a greater value than the previous one, but is still not an overestimate of the true cost to reach the goal. We can say that this heuristic dominates the previous one.

e. If we used an inadmissible heuristic in A^* tree search, could it change the completeness of the search?
Ans: No! If the heuristic function is bounded, then A^* would visit all the nodes eventually, and would find a path to the goal state if there exists one.

f. If we used an inadmissible heuristic in A^* tree search, could it change the optimality of the search?
Ans: Yes! It can make the good optimal goal look as though it is very far off, and take you to a suboptimal goal.

g. Give a general advantage that an inadmissible heuristic might have over an admissible one.
Ans: The time to solve an A^* search problem is a function of two factors: the number of nodes expanded, and the time spent per node.
An inadmissible heuristic may be faster to compute, leading to a solution that is obtained faster due to less time spent per node. It can also be a closer estimate to the actual cost function (even though at times it will overestimate!), thus expanding less nodes.

It's important to remember that we definitely lose the guarantee of optimality by using an inadmissible heuristic. But sometimes we may be okay with finding a suboptimal solution to a search problem.

2. Consider the search space below, where S is the start node and G_1, G_2, and G_3 satisfy the goal test. Arcs are labeled with the cost of traversing them and the h functions values are reported beside the graph.

   For each of the following search strategies, indicate which goal state is reached (if any) and list, in order, all the states popped off of the list. When all else is equal (i.e., to break ties), nodes should be placed in the list in alphabetical order.

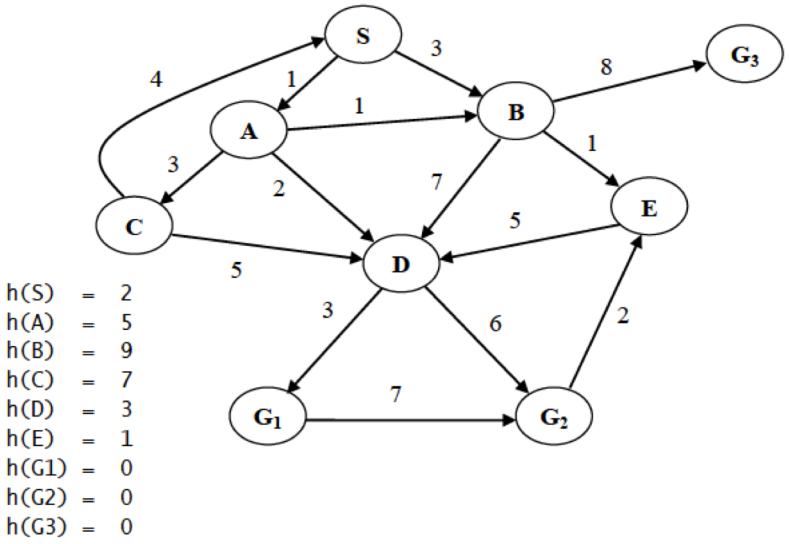

Figure 2:

   a. Breadth First

   Expands S, A, B. Finds G3. Note that BFS has a special policy for expansion as explained in teh text book – it is the only algorithm among these that performs the goal test before inserting to *open*, not after removing from *open*. We will also accept the answer (S,A,B,C,D,E,G3; finds G3).

   b. Depth First - Expands S, A, C, D, G1; find G1.

   c. Iterative Deepening - Expands S— S, A, B— S, A, C, D, B, E, G3 finds G3

   d. Uniform Cost - Expands S, A, B, D, E, C G1; finds G1

   e. Best-first (using f = h) - Expands S, A, D, G1; finds G1

   f. Best-first (using f = g + h) - Expands S, A, D, G1; finds G1

   g. Beam Search (with beam width = 2 and f = h) - Expands S, A, D, G1; finds G1

   h. Hill Climbing (using the h function only) - Expands S, finds no neighbors with lower h value, terminates. Note that hill-climbing fails because the search starts in a local minimum of h.

   i. Is this h function admissible? Explain you answer. What impact did this have on each of parts e-h above? Verbal answers are fine; you do not need to fix h and redo the searches in these parts.

   h is not admissible, because in at least one case, it overestimates the shortest path from a node to the goal. For example h(B) = 9 and yet CostofPath(B,G3) = 8. This makes it overestimate the distance and hence is not admissible.

3. Question 3.6 Check every part. We will discuss (b) in class – Give a complete formulation for each of the following. Choose a formulation that is precise enough to be implemented. (b) A 3-foot tall monkey is in a room where some bananas are suspended from the 8-foot ceiling. He would like to get the bananas. The room contains two stackable, movable, climbable, 3 foot high crates.

Ans: Initial state: As described in the text.
Goal test: Monkey has bananas.
Successor function: Hop on crate; Hop off crate; Push crate from one spot to another; Walk from one spot to another; grab bananas (if standing on crate).
Cost function: Number of actions.

4. Question 3.11 What is the difference between a world state, a state description and a search node? Why is this distinction useful?

   Ans: A world state is how reality is or could be. In one world state were in Arad, in another were in Bucharest. The world state also includes which street were on, whats currently on the radio, and the price of tea in China. A state description is an agents internal description of a world state. Examples are *In(Arad)* and *In(Bucharest)*. These descriptions are necessarily approximate, recording only some aspect of the state. We need to distinguish between world states and state descriptions because state description are lossy abstractions of the world state, because the agent could be mistaken about how the world is, because the agent might want to imagine things that are not true but it could make true, and because the agent cares about the world not its internal representation of it.

   Search nodes are generated during search, representing a state the search process knows how to reach. They contain additional information aside from the state description, such as the sequence of actions used to reach this state. This distinction is useful because we may generate different search nodes which have the same state, and because search nodes contain more information than a state representation.

5. Question 4.1 Give the name of the algorithm that results from each of the following special cases:

   - Local beam search with $k = 1$ is hill-climbing search.
   - Local beam search with one initial state and no limit on the number of states retained, resembles breadth-first search in that it adds one complete layer of nodes before adding the next layer. Starting from one state, the algorithm would be essentially identical to breadth-first search except that each layer is generated all at once.
   - Simulated annealing with $T = 0$ at all times: ignoring the fact that the termination step would be triggered immediately, the search would be identical to first-choice hill climbing because every downward successor would be rejected with probability 1.
   - Simulated annealing with $T = \infty$ at all times is a random-walk search: it always accepts a new state.
   - Genetic algorithm with population size $N = 1$: if the population size is 1, then the two selected parents will be the same individual; crossover yields an exact copy of the individual; then there is a small chance of mutation. Thus, the algorithm executes a random walk in the space of individuals.

3 Adversarial Search

1. Question 5.7 Prove the following assertion: For every game tree, the utility obtained by MAX using minimax discussions against a suboptimal MIN will never be lower than the utility obtained playing against an optimal MIN. Can you come up with a game tree in which MAX can do still better using a *suboptimal* strategy against a suboptimal MIN?

   Ans: Refer to homework #2.

2. Question 5.12 Describe how the minimax and alpha-beta algorithms change for two-player, non-zero sum games in which player has a distinct utility function and both utility functions are known to both players. If there are no constraints on the two terminal utilities, is it possible for any node to be pruned by alpha-beta? What if the player's utility functions on any state differ by at most a constant k, making the game almost cooperative?

   Ans: Refer to homework #2.

3. The standard Minimax algorithm calculates worst-case values in a zero-sum two player game, i.e. a game in which for all terminal states s, the utilities for players A (MAX) and B (MIN) obey $U_A(s) + U_B(s) = 0$. In the zero sum case, we know that $U_A(s) = -U_B(s)$ and so we can think of player B as simply minimizing $U_A(s)$. In this problem, you will consider the non zero-sum generalization in which the sum of the two

players' utilities are not necessarily zero. Because player A's utility no longer determines player B's utility exactly, the leaf utilities are written as pairs (U_A, U_B), with the

first and second component indicating the utility of that leaf to A and B respectively. In this generalized setting, A seeks to maximize U_A, the

first component, while B seeks to maximize U_B, the second component.

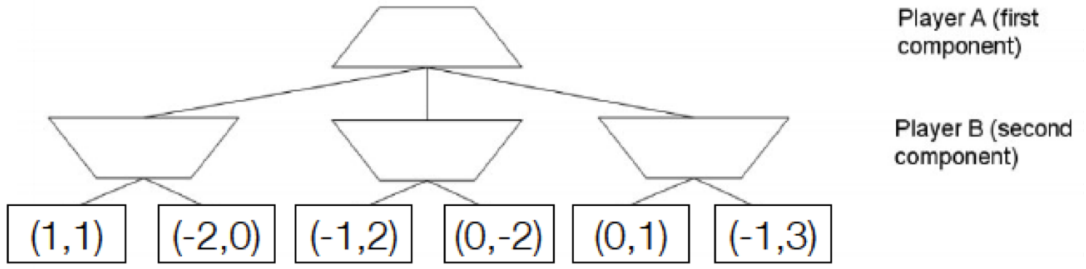

Figure 3:

- Propagate the terminal utility pairs up the tree presented in Figure 5 using the appropriate generalization of the minimax algorithm on this game tree. Fill in the values (as pairs) at each of the internal node. Assume that each player maximizes their own utility.

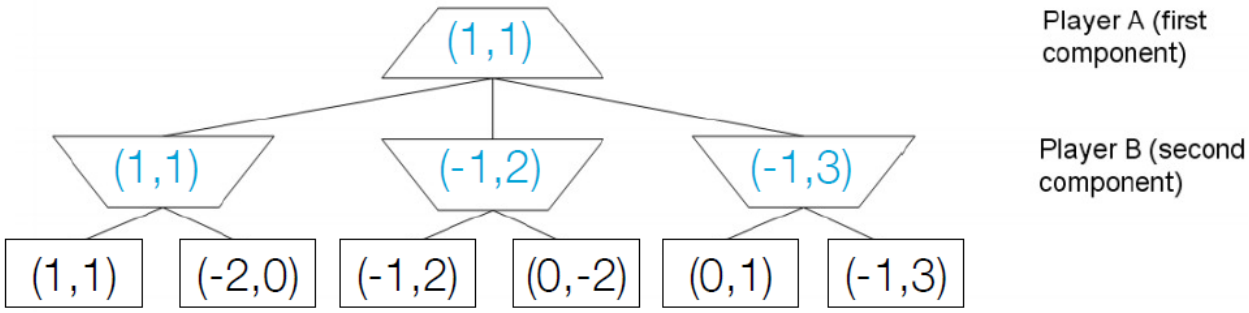

Figure 4:

- Briefly explain why no alpha-beta style pruning is possible in the general non-zero sum case. Hint: think first about the case where $U_A(s) = U_B(s)$ for all nodes.

  Ans: The values that the first and second player are trying to maximize are independent, so we no longer have situations where we know that one player will never let the other player down a particular branch of the game tree.

  For instance, in the case where $U_A = U_B$, the problem reduces to searching for the max-valued leaf, which could appear anywhere in the tree.

4 Propositional Logic

1. Represent each of these sentences in propositional logic.

- If one has a fever, one should not go to work. $F \rightarrow \neg W$
- It is hot but not sunny. $H \wedge \neg S1$
- One should not bike on highways. $H \rightarrow \neg B$

- Do not drink or text when driving. $C \rightarrow \neg(D \wedge T)$

2. For each of the formulae, use truth tables to show whether each of them is valid, satisfiable or unsatisfiable

a. $(P \rightarrow Q) \wedge (P \rightarrow \neg Q)$

P	Q	$P \rightarrow Q$	$P \rightarrow \neg Q$	WFF
F	F	T	T	T
F	T	T	T	T
T	F	T	F	F
T	T	F	F	F

Since the formula is satisfied in some assignments, it is **satisfiable**.

b. $(\neg P \vee \neg Q) \wedge (\neg Q \vee \neg R) \wedge (\neg R \vee \neg P) \wedge P \wedge Q$

P	Q	R	$\neg P \vee \neg Q$	$\neg Q \vee \neg R$	$\neg R \vee \neg P$	WFF
F	F	F	T	T	T	F
F	F	T	T	T	T	F
F	T	F	T	T	T	F
F	T	T	T	F	T	F
T	F	F	T	T	T	F
T	F	T	T	T	F	F
T	T	F	F	T	T	F
T	T	T	F	F	F	F

Since the formula is not satisfied for any assignment, we say that the formula is **unsatisfiable**. Note that the clauses $(\neg P \vee \neg Q)$, P and Q alone cannot be all satisfied by any assignment.

c. $((P \rightarrow Q) \wedge (Q \rightarrow R)) \leftrightarrow (P \rightarrow R)$

P	Q	R	$((P \rightarrow Q)$	$(Q \rightarrow R))$	$(P \rightarrow R)$	WFF
F	F	F	T	T	T	T
F	F	T	T	T	T	T
F	T	F	T	F	T	F
F	T	T	T	T	T	T
T	F	F	F	T	F	T
T	F	T	F	T	T	F
T	T	F	T	F	F	T
T	T	T	T	T	T	T

The formula is satisfied by some but not all assignments. Hence it is **satisfiable**.

d. $((P \rightarrow Q) \rightarrow (Q \rightarrow R)) \leftrightarrow (P \rightarrow R)$
As can be seen from Table 1, the formula is satisfied by some but not all assignments. Hence it is **satisfiable**.

3. Put each of the following in CNF form

- $(P \rightarrow Q) \wedge (R \vee S)$

$$(\neg P \vee Q) \wedge (R \vee S)$$

- $(P \wedge Q \rightarrow (X \vee Y)) \vee (R \wedge S)$

Note that $\wedge$ has a higher precedence than $\rightarrow$. So you should assume $P \wedge Q$ as the precondition of the implication. So the above must be rewritten as follows:

$$\begin{aligned} & ((P \wedge Q) \rightarrow (X \vee Y)) \vee (R \wedge S) \\ = & (\neg(P \wedge Q) \vee (X \vee Y)) \vee (R \wedge S) \\ = & (\neg P \vee \neg Q \vee X \vee Y) \vee (R \wedge S) \\ = & (\neg P \vee \neg Q \vee X \vee Y \vee R) \wedge (\neg P \vee \neg Q \vee X \vee Y \vee S) \end{aligned} \quad (1)$$

P	Q	R	$((P \to Q)$	$(Q \to R))$	$(P \to R)$	WFF
F	F	F	T	T	T	T
F	F	T	T	T	T	T
F	T	F	T	F	T	F
F	T	T	T	T	T	T
T	F	F	F	T	F	F
T	F	T	F	T	T	T
T	T	F	T	F	F	T
T	T	T	T	T	T	T

Table 1: Part d truth table

- $((P \to Q) \to \neg X) \leftrightarrow (A \wedge B)$

$$
\begin{aligned}
&((P \to Q) \to \neg X) \leftrightarrow (A \wedge B) \\
&= [((P \to Q) \to \neg X) \leftarrow (A \wedge B)] \wedge [(A \wedge B) \leftrightarrow ((P \to Q) \to \neg X)] \\
&= [((\neg P \vee Q) \to \neg X) \leftarrow (A \wedge B)] \wedge [(A \wedge B) \leftrightarrow ((\neg P \vee Q) \to \neg X)] \\
&= [(\neg(\neg P \vee Q) \vee \neg X) \leftarrow (A \wedge B)] \wedge [(A \wedge B) \leftrightarrow (\neg(\neg P \vee Q) \vee \neg X)] \\
&= [\neg(\neg(\neg P \vee Q) \vee \neg X) \vee (A \wedge B)] \wedge [\neg(A \wedge B) \vee (\neg(\neg P \vee Q) \vee \neg X)] \\
&= [((\neg P \vee Q) \wedge X) \vee (A \wedge B)] \wedge [\neg A \vee \neg B \vee (P \wedge \neg Q) \vee \neg X] \text{DeMorgan's Law} \\
&= (((\neg P \vee Q) \wedge X) \vee A) \wedge (((\neg P \vee Q) \wedge X) \vee B) \wedge [\neg A \vee \neg B \vee (P \wedge \neg Q) \vee \neg X] \text{distribution of } \vee \text{ over } \wedge \\
&= (\neg P \vee Q \vee A) \wedge (X \vee A) \wedge (\neg P \vee Q \vee B) \wedge (X \vee B) \wedge [\neg A \vee \neg B \vee (P \wedge \neg Q) \vee \neg X] \text{distribution of } \vee \text{ over } \wedge \\
&= (\neg P \vee Q \vee A) \wedge (X \vee A) \wedge (\neg P \vee Q \vee B) \wedge (X \vee B) \; wedge (\neg A \vee \neg B \vee \neg X \vee P) \wedge (\neg A \vee \neg B \vee \neg X \vee \neg Q)
\end{aligned}
$$

4. Given,

   1. $P \wedge Q \to R$
   2. $A \wedge R \to B$
   3. $Q \wedge \neg C \to A$
   4. Q
   5. $\neg(\neg P)$
   6. $\neg C$

   Show: $R \wedge A$

   Prove the above concept in two ways: first a normal deduction – let us call it "natural deduction" and second method is using *only* resolution rule.

   - Two inference rules that we will use are AND introduction and Modus Ponens. An AND introduction simply allows us to put a $\wedge$ between two formulas that are known to be true. Modus Ponens means that if we have an implication, and we know that the precondition of the implication holds, then we can deduce that the postcondition of the implication also holds. (In notation, if we know that $A \wedge B \to C$ and we know $A \wedge B$, we can deduce C) Table 2 presents the natural deduction steps needed to show that $R \wedge A$ holds.
   - For using resolution we need to convert all our clauses:
     1. $\neg P \vee \neg Q \vee R$
     2. $\neg A \vee \neg R \vee B$
     3. $\neg Q \vee C \vee A$
     4. Q
     5. P
     6. $\neg C$

Number	WFF	Justification
7	P	Double negation and elimination on 5
8	$P \wedge Q$	AND introduction on 4 and 7
9	R	Modus Ponens on 1 and 8
10	$Q \wedge \neg C$	AND introduction on 4 and 6
11	A	Modus Ponens on 5 and 10
12	$R \wedge A$	AND introduction on 9 and 11

Table 2: Natural derivation of $R \wedge A$

These converted formulas represent our knowledge base. In a proof by contradiction, we want to show that assuming our knowledge base is true and our query is false produces a contradiction. This contradiction will prove that our query must be true, given our knowledge base. In this case, our negated query is : $\neg(R \wedge A) = \neg R \vee \neg A$. We start with all clauses in our knowledge base plus the negated query, and begin to apply the resolution rule. Figure **??** shows the steps involved in the proof.

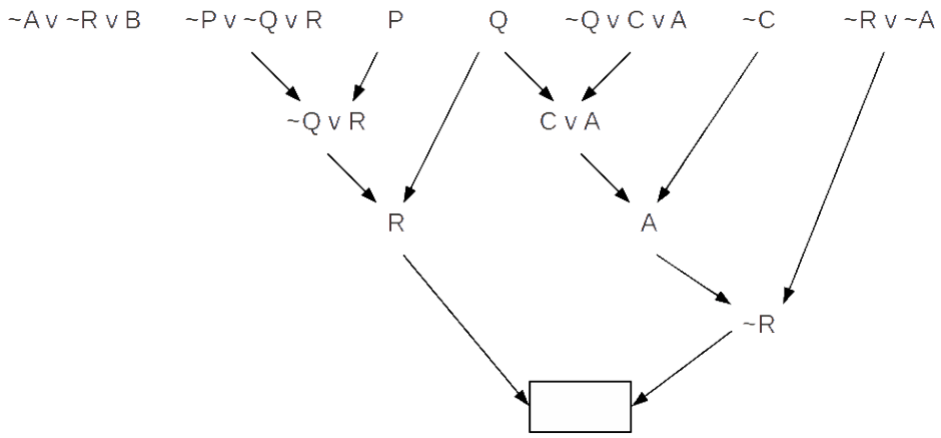

Figure 5: Steps involved in resolution

5. Consider the following sentence,

$$[(Food \rightarrow Party) \wedge (Drinks \rightarrow Party)] \rightarrow [(Food \wedge Drinks) \rightarrow Party]$$

- Determine, using enumerate whether this sentence is valid, satisfiable or invalid.
  Ans A simple truth table has eight rows, and shows that the sentence is true for all models and hence valid.
- Convert the left hand and right hand sides of the implication into CNF, showing each step and explain how the results confirm to the previous answer.
  Ans For the left hand side we have

$$\begin{aligned}(Food \rightarrow Party) \vee (Drinks \rightarrow Party)\\ (\neg Food \vee Party) \vee (\neg Drinks \vee Party)\\ (\neg Food \vee Party \vee \neg Drinks \vee Party)\\ (\neg Food \vee \neg Drinks \vee Party)\end{aligned}$$

  For the right hand side, we have,

$$\begin{aligned}(Food \wedge Drinks) \rightarrow Party\\ \neg(Food \wedge Drinks) \vee Party\\ (\neg Food \vee \neg Drinks) \vee Party\\ (\neg Food \vee \neg Drinks \vee Party)\end{aligned}$$

  The two sides are identical in a CNF and is of the form $P \rightarrow P$, which is valid for any P.

- Prove your answer to the first part using resolution.
  Ans To prove this, we should use the result from the above CNF

  $(\neg Food \vee \neg Drinks \vee Party) \wedge Food \wedge Drinks \wedge \neg Party$
  Each of the three unit clauses resolves in turn against the first clause, leaving an empty clause.

6. Question 7.20. Convert the following set of sentences to clausal form.

   1. $A \leftrightarrow (B \vee E)$
   2. $E \rightarrow D$
   3. $C \wedge D \rightarrow \neg B$
   4. $E \rightarrow B$
   5. $B \rightarrow F$
   6. $B \rightarrow C$

   Give a trace of the execution of DPLL on the conjunction of these clauses. (Ignore this part).
   Ans: DPLL trace is easy to obtain. We can ignore it for this part.

   1. $(\neg A \vee B \vee E) \wedge (\neg B \vee A) \wedge (\neg E \vee A)$
   2. $\neg E \vee D$
   3. $\neg C \vee D \vee \neg B$
   4. $\neg E \vee B$
   5. $\neg B \vee F$
   6. $\neg B \vee C$