James Hahn
Project 3
CS1571 – Diane Litman

1. Refer to the following truth table. The value 1 indicates true. The value 0 indicates false.

A	**B**	**C**	**¬C**	**(A or C)**		**(B or ¬C)**	**(A or C) AND (B or ¬C)**	**A or B**
0	0	0	1	0		1	0	0
0	1	0	1	0		1	0	1
0	0	1	0	1		0	0	0
0	1	1	0	1		1	1	1
1	0	0	1	1		1	1	1
1	1	0	1	1		1	1	1
1	0	1	0	1		0	0	1
1	1	1	0	1		1	1	1

As we can see in this table, when ((A or C) and (B or ¬C)), our knowledge base, is true, the statement (A or B) is also true.

2.
a) We need to convert ((P and Q) or (Q and R)) into a CNF formula with the following rules:

R -> S
S -> P
(P and Q) -> G

Goal: G

Proof shown below:

((P and Q) or (Q and R)) and (R -> S) and (S -> P) and ((P and Q) -> G)
((P and Q) or (Q and R)) or (¬R or S) and (¬S or P) and (¬(P and Q) or G)
(Q and (P or R)) or (¬R or S) and (¬S or P) and ((¬P or ¬Q) or G)
Q and (P or R) and (¬R or S) and (¬S or P) and (¬P or ¬Q or G)

As we can see, this is a CNF formula because it is a series of clauses (only containing ORs) being ANDed together.

b) We must use proof by refutation and resolution as the single inference rule to show the resolution proof proves or disproves the goal. Consider the statement formed from part a.

To prove or disprove the goal, let’s try to prove the goal G so our assumption is ¬G.

Prove G

1. Q
2. P or R
3. ¬R or S
4. ¬S or P
5. ¬P or ¬Q or G
6. ¬G **(our assumption)**
7. ¬P or G **(resolve 1 and 5)**
8. ¬R or P (**resolve 3 and 4)**
9. P **(resolve 2 and 8)**
10. G (**resolve 7 and 9)**
11. Empty **(resolve 6 and 10)**

Due to our resolution of G and ¬G being empty, our initial assumption of ¬G must be false. Therefore, G is true.

3.
We are given several pieces of information in the problem:
GameX says it is criminal for a programmer to provide emulators to people.
My friends don't have a GameX, but they use software EMULATOR1 that runs GameX games on their PC, which is written by SuperProgrammer, who is a programmer.

The predicates can be found here:

programmer(x) = x is a programmer (domain = all people)
emulator(x) = x is an emulator (domain = all pieces of software)
person(x) = x is a person
gives(x, y, z) = x gives y to z (domain = all people for x, all items for y, all people for z)
friend(x) = x is my friend (domain = all people)
gameX(x) = x is a GameX system
owns(x, y) = x owns y
software(x) = x is a piece of software
gameXgame(x) = x is a GameX game
PC(y) = y is a PC/Personal Computer
runs(x, y, z) = software x runs y on system z (used down below to represent an emulator running a game on a specific system)

If programmer x provides person z with emulator y, then x is a criminal:
∃ x, y, z, programmer(x) and emulator(y) and person(z) and gives(x, y, z) -> criminal(x)

My friend x does not own GameX y:
∃ x, y, friend(x) and gameX(y) and ¬owns(x, y)

My friend w uses EMULATOR1 to run GameX game x on their personal computer y: ∃ w, x, y, friend(w) and software(EMULATOR1) and gameXgame(x) and PC(y) and runs(EMULATOR1, x, y)

The reason I place variable y in the existential quantifier is because friend w may have several PCs but is only running GameX game x on one of their PCs.

Finally, the provider of EMULATOR1 is SuperProgrammer to their friend x and he/she is a programmer:
∃ x, programmer(SuperProgrammer) and gives(SuperProgrammer, EMULATOR1, x)

As one can see, we can use forward chaining with objects x, y, and z, as well as software and emulators to deduce whether a programmer is a criminal. Conversely, if somebody is a criminal, we can track the logic backwards and figure out why they are a criminal (in this case, it's because they provided their friend with an emulator).

Although I don't think we have to specify this for the question, but SuperProgrammer is technically not a criminal. This is because EMULATOR1 is a piece of software, but not necessarily an emulator. A piece of software can have any name and EMULATOR1 could just so happen to be MatLab or something that's unrelated. Therefore, we are missing a small bit of information in the problem to definitively say whether SuperProgrammer is a criminal or not under GameX's standards. As it is now, SuperProgrammer is not a criminal.

To definitely say whether he/she is a criminal, I would change the software(EMULATOR1) predicate above to emulator(EMULATOR1). Then, if that returns true, since emulators are automatically pieces of software anyway, SuperProgrammer would be a criminal.

4.
a) {x|A, y|B, z|B}
b) There is no unifier. Can't unify A with B.
c) {y|John, x|John}
d) There is no unifier. Can't unify y with Father(y).

5.
a) Let's first convert each individual statement:

1. ∀ a, ¬(pass(a, History) and win(a, Lottery)) or happy(a)
2. ∀ b, c, ¬(study(b) or lucky(b)) or pass(b, c)
3. ¬study(John) and lucky(John)
4. ∀ d, ¬lucky(d) or win(d, Lottery)
5. ∃ e, wealthy(e)

After applying DeMorgan's laws, we can simplify two of our above statements (#1 and #2) and get these final five statements:

1. ∀ a, ¬pass(a, History) or ¬win(a, Lottery) or happy(a)
2. ∀ b, c, (¬study(b) or pass(b, c)) and (¬lucky(b) or pass(b, c))
3. ¬study(John) and lucky(John)
4. ∀ d, ¬lucky(d) or win(d, Lottery)
5. ∃ e, wealthy(e)

Finally, we have:

∀ a, b, c, d, ∃ e, (¬pass(a, History) or ¬win(a, Lottery) or happy(a)) and (¬study(b) or pass(b, c)) and (¬lucky(b) or pass(b, c)) and ¬study(John) and lucky(John) and (¬lucky(d) or win(d, Lottery)) and wealthy(e)

As you can see, we have seven clauses, only containing ORs, that are all ANDed together. Therefore, we have successfully converted the statement into a CNF formula.

b) We must use proof by refutation and resolution as the single inference rule to show John is happy. Consider the statement formed from part a:

∀ a, b, c, d, ∃ e, (¬pass(a, History) or ¬win(a, Lottery) or happy(a)) and (¬study(b) or pass(b, c)) and (¬lucky(b) or pass(b, c)) and ¬study(John) and lucky(John) and (¬lucky(d) or win(d, Lottery)) and wealthy(e)

Prove happy(John)

1. ¬happy(John) **(our assumption)**
2. ¬pass(a, History) or ¬win(a, Lottery) or happy(a)
3. ¬study(b) or pass(b, c)
4. ¬lucky(b) or pass(b, c)
5. ¬study(John)
6. lucky(John)
7. ¬lucky(d) or win(d, Lottery)
8. wealthy(e)
9. ¬pass(John, History) or ¬win(John, Lottery) **(resolve with 1 and 2)** (unify {a|John})
10. pass(John, c) **(resolve with 4 and 6)**
11. win(John, Lottery) **(resolve with 6 and 7)** (unify {d|John})
12. ¬win(John, Lottery) or happy(John) **(resolve with 2 and 10)** (unify {a|John, c|History})
13. happy(John) **(resolve with 11 and 12)**
14. Empty clause **(resolve with 13 and 1)**

Since we have resolved to an empty clause, ¬happy(John) must be false, so we have proven happy(John) by refutation.