Afshin Jamali
CSC 480
Assignment 3

Part 1: Proposal Logic

Facts about this world:

- **(1) says(A,S) /\ knave(A) => ~S**
- **(2) says(A,S) /\ knight(A) => S**
- **(3) ~knight(A) => knave(A)**
- **(4) ~knave(A) => knight(A)**

Q1:

- Suppose B is a Knight
  - Knight(B)
  - Says(B, ~(Knave(B) /\ Knave(A)))
  - By (2) and MP we can conclude: ~Knave(B) /\ ~Knave(A)
  - By AE: ~Knave(B)
  - there is no contradiction here
  - But we need to prove that: ~Knave(A) for Knight(B) to be true
- What is A?
  - We know by (2) and from above that Says(A,S) must be true for Knight(B) to be true
  - Says(A, Knave(B))
  - This is a contradiction, so our assumption that Knight(B) was false
  - By (3), and the resolution rule we conclude Knave(B) and Knight(A)

Q2:

- Suppose A is a Knight
  - Knight(A)
  - Says(A, Knight(A) /\ Knight(B))
  - By AE: Knight(A)
  - No contradictions here, but Knight(B) must also be true for Knight(A) to be true
- Is B a Knight?
  - We know by (2) and from above that Says(B,S) must be true for Knight(A) to be true
  - Says(B, Knight(B) /\ Knave(A))
  - By (2), MP, and AE we conclude: Knight(B)
  - This is a contradiction for Says(A,S)
  - By (3) and the resolution rule we conclude Knave(A)

Q3:

Answer is yes.

We know that: (Freedom(Left Road) \/ Freedom(Right Road)) and KB

What we also know which is critical (new fact): **(5) Knight(A) \/ Knave(A)**

Question can be: <u>Would the other member answer yes if asked that the road to freedom is the left road?</u>

- Suppose A is a Knight and Freedom(Left Road)
  - Knight(A)
  - By (2) and (5) and KB, we can conclude: ~Freedom(Left Road)
  - Therefore the Knight would answer no
  - The knight will always tell the truth and inform us that the response from the Knave would be no
- Now suppose A is a Knave with same setting: Freedom(Left Road)
  - Knave(A)
  - By (1) and (5) and KB, we can conclude: ~Freedom(Left Road)
  - Therefore the Knave will lie about the Knight and say no
  - The knave will also not tell the truth about the Knight
  - We know that regardless if A is Knight or A is Knave the response is the same
  - Based on this tautology we can conclude: Response(A,S) ←→ Response(B,S)
  - ~(Response(A,S) /\ Response(B,S))
  - By DeMorgan's law: ~Response(A,S) \/ ~Response(B,S)
  - Hence, the opposite of what A or B says will be the true road to freedom

<u>Part 2: FOL</u>

Q4:

1. $\forall$x,y Creature(x) /\ Creature(y) /\ CanEat(x,y) → CanClobber(x,y)
2. $\forall$x, $\exists$y Monster(x) /\ Creature(y) /\ $\neg$Monster(y) → CanEat(x,y)
3. $\forall$x, y, z CanClobber(x, y) /\ CanClobber(y, z) → CanClobber(x, z)
4. $\forall$x$\forall$y IsOgre(x) /\ IsDwarf(y) → CanEat(x, y)
5. $\forall$x $\forall$y IsDwarf(x) /\ IsGoblin(y) → CanClobber(x,y)
6. $\forall$x IsGoblin(x) /\ IsMonster(x)

Q5:

1.
   a. $\forall$x,y Creature(x) /\ Creature(y) /\ CanEat(x,y) → CanClobber(x,y)

b. Eliminate → and drop universals: ¬(Creature(x) /\ Creature(y) /\ CanEat(x,y))) \/ CanClobber(x,y)
c. Move in Negation: ¬Creature(x) \/ ¬Creature(y) \/ ¬CanEat(x, y) \/ CanClobber(x,y)
d. Distribute /\ over \/: (¬Creature(x) \/ ¬Creature(y)) /\ (¬Creature(x) \/ ¬CanEat(x, y)) /\ (¬Creature(x) \/ CanClobber(x,y))
e. Split Conjunction: (¬Creature(x) \/ ¬Creature(y)), (¬Creature(x) \/ ¬CanEat(x, y)), (¬Creature(x) \/ CanClobber(x,y))

2.
   a. ∀x, ∃y Monster(x) /\ Creature(y) /\ ¬Monster(y) → CanEat(x,y)
   b. ¬(Monster(x) /\ Creature(sk(y) /\ ¬Monster(sk(y))) \/ CanEat(x,sk(y))
   c. ¬Monster(x) \/ ¬Creature(sk(y)) \/ Monster(Sk(y)) \/ CanEat(x,sk(y))
   d. (¬Monster(x) \/ ¬Creature(sk(y)) /\ (¬Monster(x) \/ Monster(sk(y)) /\ (¬Monster(x) \/ CanEat(x, sk(y))
   e. (¬Monster(x) \/ ¬Creature(sk(y)), (¬Monster(x) \/ Monster(sk(y)), (¬Monster(x) \/ CanEat(x, sk(y))

3.
   a. ∀x, y, z CanClobber(x, y) /\ CanClobber(y, z) → CanClobber(x, z)
   b. ¬(CanClobber(x, y) /\ CanClobber(y, z)) \/ CanClobber(x, z)
   c. (¬CanClobber(x, y) \/ ¬CanClobber(y, z)) \/ CanClobber(x, z)
   d. (¬CanClobber(x, y) \/ ¬CanClobber(y, z)) /\ (¬CanClobber(x, y) \/ CanClobber(x, z))
   e. (¬CanClobber(x, y) \/ ¬CanClobber(y, z)), (¬CanClobber(x, y) \/ CanClobber(x, z))

4.
   a. ∀x∀y IsOgre(x) /\ IsDwarf(y) → CanEat(x, y)
   b. ¬(IsOgre (x) /\ IsDwarf(y)) \/ CanEat(x, y)
   c. ¬ IsOgre (x) \/ ¬IsDwarf(y) \/ CanEat(x,y)
   d. (¬IsOgre (x) \/ ¬IsDrwarf) /\ (¬IsOgre (x) \/ CanEat(x,y))
   e. (¬IsOgre (x) \/ ¬IsDrwarf), (¬IsOgre (x) \/ CanEat(x,y))

5.
   a. ∀x ∀y IsDwarf(x) /\ IsGoblin(y) → CanClobber(x,y)
   b. ¬(IsDwarf(x) /\ IsGoblin(y)) \/ CanClobber(x,y)
   c. (¬IsDwarf(x) \/ ¬IsGoblin(y)) \/ CanClobber(x,y)
   d. (¬IsDwarf(x) \/ ¬IsGoblin(y)) /\ (¬IsDwarf(x) \/ CanClobber(x,y))
   e. (¬IsDwarf(x) \/ ¬IsGoblin(y)), (¬IsDwarf(x) \/ CanClobber(x,y))

6.
   a. ∀x IsGoblin(x) /\ IsMonster(x)
   b. IsGoblin(x), IsMonster(x)

Q6:

Knowledge Base:

1. (¬Creature(x) \/ ¬Creature(y))

2. (¬Creature(x) \/ ¬CanEat(x, y))
3. (¬Creature(x) \/ CanClobber(x,y))
4. (¬Monster(x) \/ ¬Creature(sk(y))
5. (¬Monster(x) \/ Monster(sk(y))
6. (¬Monster(x) \/ CanEat(x, sk(y))
7. (¬CanClobber(x, y) \/ ¬CanClobber(y, z))
8. (¬CanClobber(x, y) \/ CanClobber(x, z))
9. (¬IsOgre (x) \/ ¬IsDrwarf)
10. (¬IsOgre (x) \/ CanEat(x,y))
11. (¬IsDwarf(x) \/ ¬IsGoblin(y))
12. (¬IsDwarf(x) \/ CanClobber(x,y))
13. IsGoblin(x)
14. Monster(x)
15. isDwarf(x)

Prove: CanClobber(Ogre, Goblin) /\ IsMonster(goblin)

16. ¬CanClobber(x,y) /\ ¬IsMonster(goblin)
17. (16, 12) ¬IsDwarf /\ ¬IsMonster(goblin) y = goblin
18. (17, 5) ¬IsMonster(x) /\ ¬IsDwarf(x)
19. (18, 15) ¬Monster(x)
20. (18, 14) NIL

Therefore ¬(¬CanClobber(x,y))

Q7:

16. ¬CanEat(Ogre, dwarf)   x = ogre, y = dwarf
17. (16, 6) ¬Monster(x)
18. (17,14) NIL

Therefore ¬(¬ CanEat (x,y))

Part 3: Prog

Q8:

Q8a. true
Q8b. X = mia
Q8c. X = mia
Q8d. false
Q8e. X = Vincent, X = marsellus
Q8f. X = Vincent, X = Marsellus

Q8g. X = honey_bunny

Q9:

Code:

```
car(bmw).
car(civic).
motorcycle(harley).
fast(bmw).
slow(civic).
fast(harley).
fast(theFlash).

fun(X):-  car(X),  fast(X).
fun(X):-  motorcycle(X),  fast(X).
```

Query:

1. fun(bmw).   //true, false
2. fun(Harley).   //true
3. fun(civic).   //fales
4. fun(theFlash).   //false

Q10:

Code:

```
creature(dwarf).
creature(ogre).
monster(goblin).
eats(ogre,dwarf).
eats(dwarf, goblin).
creature(X):-  monster(X).
eats(X, otherCreatures):-  monster(X).
clobbers(X,Y):-  eats(X,Y),  creature(X),  creature(Y).
clobbers(X,Y):-  clobbers(X,Z),  clobbers(Z,Y).
```

Query:

1. Clobbers(ogre, goblin).   //true
2. Eats(ogre,X).   //X = dwarf