[1] Describe with examples, the three inference rules involving quantifiers

The three new inference rules are as follows:

1- **Universal Instantiation :** For any sentence *a*, variable *v*, and ground term g (a term without variables):

$$\frac{\forall v \quad \alpha}{\text{SUBST}(\{v/g\},\alpha)}$$

For example, from $\forall x$ *Likes(x, IceCream)*, we can use the substitution *{x/Ben}* and infer *Likes(Ben, IceCream).*

2- **Existential Instantiation:** For any sentence *a*, variable *v* , and constant symbol *k* that does not appear elsewhere in the knowledge base:

$$\frac{\exists v \quad \alpha}{\text{SUBST}(\{v/k\},\alpha)}$$

For example, from ∃ *x Kill(x, Victim)*, we can infer *Kill(Murderer, Victim)*, as long as *Murderer* does not appear elsewhere in the knowledge base.

3- **Existential Introduction:** For any sentence *a*, variable *v* that does not occur in *a*, and ground term *g* that does occur in *a*:

$$\frac{\alpha}{\exists v \quad \text{SUBST}(\{g/v\},\alpha)}$$

For example, from *Likes(Jerry, IceCream)* we can infer ∃ *x Likes(x, IceCream).*

[2] Write the syntax of **Generalized Modus Ponens** and **Generalized Resolution**

Generalized Modus Ponens:

For atomic sentences p_i, p_i', and q, where there is a substitution θ such that SUBST(θ, p_i') = SUBST(θ, p_i), for all *i:*

$$\frac{p_1', p_2', \ldots p_n', (p_1 \wedge p_2 \wedge \ldots \wedge p_n \Rightarrow q)}{\text{SUBST}(\theta, q)}$$

Generalized Resolution

For literals p_j, and q_k where UNIFY(p_j , q_k) = *0:*

$$\frac{\begin{array}{c} p_1 \vee \ldots p_j \ldots \vee p_m \\ q_1 \vee \ldots q_k \ldots \vee q_n \end{array}}{SUBST\left(\theta, p_1 \vee \ldots p_{j-1} \vee p_{j+1} \ldots \vee p_m \vee q_1 \vee \ldots q_{k-1} \vee q_{k+1} \ldots \vee q_n\right)}$$

[3] define the job of *Unify* routine, then find out the result of unifying :

a) UNIFY (Knows(John, x), Knows(John, Jane))
b) UNIFY (Knows(John, x), Knows(y, Leonid))
c) UNIFY (Knows(John, x), Knows(y, Mother (y)))
d) UNIFY (Knows(John, x), Knows(x, Elizabeth))

The job of the unification routine, UNIFY, is to take two atomic sentences *p* and *q* and return a substitution that would make *p* and *q* look the same.
If there is no such substitution, then UNIFY should return *fail*. Formally,

$$\text{UNIFY}(p, q) = \theta \text{ where SUBST}(\theta, p) = \text{SUBST}(\theta, q)$$

θ is called the **unifier** of the two sentences.

Result of unification:

a) {x/Jane}
b) {x/Leonid, y/John}
c) {y/John, x/Mother(John)}
d) fail

[4] Represent the following sentences in first-order logic, then in CNF

1. Animals can outrun any animals that they can eat.
2. Carnivores eat other animals.
3. Outrunning is transitive; if x can outrun y and y can outrun z, then x can outrun z.
4. Lions eat zebras.
5. Zebras can outrun dogs.
6. Dogs are carnivores.

FOL expressions:

1. $\forall$ x, y eats(x, y) $\Rightarrow$ outruns(x, y)
2. $\forall$ x carnivorous(x) $\Rightarrow$ $\exists$ y eats(x, y)
3. $\forall$ x, y, z outruns(x, y) $\wedge$ outruns(y, z) $\Rightarrow$ outruns(x, z)
4. eats(Lions, Zebras)
5. outruns(Zebras, dogs)
6. carnivorous(Dogs)

CNF expressions:

1. $\neg$ eats(x1, y1) $\vee$ outruns(x1, y1)
2. $\neg$ carnivorous(x2) $\vee$ eats(x2, food(x2))
3. $\neg$ outruns(x3, y2) $\vee$ $\neg$ outruns(y2, z1) $\vee$ outruns(x3, z1)
4. eats(Lions, Zebras)
5. outruns(Zebras, dogs)
6. carnivorous(Dogs)

[5] FORWARD-CHAIN algorithm is based on FIND-AND-INFER procedure and composition.

a) Write FIND-AND-INFER procedure.

b) Define the idea of a **composition** of substitutions.

a)

procedure FIND-AND-INFER(*KB*, *premises*, *conclusion*, θ)
    if *premises* = [ ] **then**
        FORWARD-CHAIN(*KB*, SUBST(θ, *conclusion*))
    else for each *p'* **in** *KB* such that UNIFY (*p'*, SUBST(θ, FIRST(*premises*))) = θ_2 **do**
        FIND-AND-INFER (*KB*, REST*(premises)*, *conclusion*, COMPOSE(θ, θ_2))
end

b)

COMPOSE(θ, θ_2) is the substitution whose effect is identical to the effect of applying each substitution in turn. That is,

SUBST(COMPOSE(θ_1, θ_2) , p) = SUBST(θ_2,SUBST(θ_1,p))

[6] Using the following facts

1. Marcus was a man.
2. Marcus was a Pompeian.
3. All Pompeians were Romans.
4. Caesar was a ruler.
5. All Romans were either loyal to Caesr or hated him.
6. Everyone is loyal to someone.
7. People only try to assassinate rulers they are not loyal to.
8. Marcus tried to assassinate Caesar.

Answer the question "Did Marcus hate Caesar".

Hint : Write CNF sentences, then try to prove that Marcus hate Caesar

The answer:

The facts:

1- Man (Marcus)

2- Pompeian (Marcus)

3- $\neg$Pompeian (X1) $\vee$ Roman (X1)

4- Ruler (Caeser)

5- $\neg$Roman (X2) $\vee$ loyalto(X2,Caeser) $\vee$ hate(X2,Caeser)

6- loyalto(X3,f1(X3))

7- $\neg$man(X4) $\vee$ $\neg$ruler(Y1) $\vee$ $\neg$tryassassinate(X4,Y1) $\vee$ loyalto(X4,Y1)

8- tryassassinate(Marcus,Caeser)

9- persecute(X,Y) $\rightarrow$ hate(Y,X)

10- hate(X,Y) $\rightarrow$ persecute(Y,X)

Using resolution: hate(Marcus,Caeser)

$\neg$ hate (Marcus,Caesar)   5

Marcus / X2

3    $\neg$Roman (Marcus) $\vee$ loyalto(Marcus,Caeser)

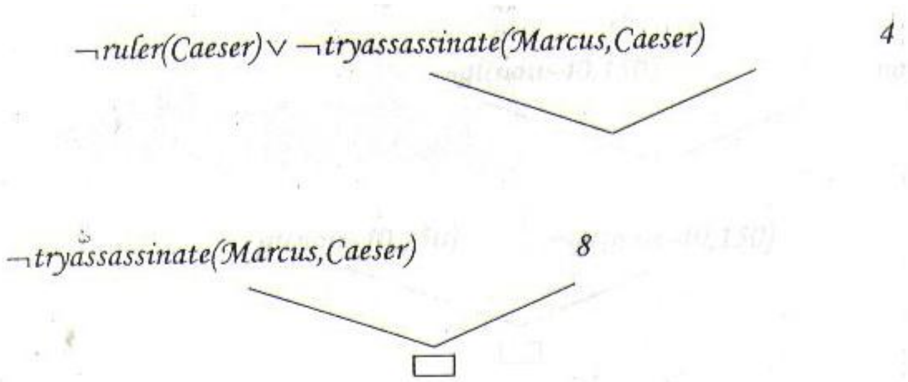
Marcus / X1

$\neg$Pompeian (X1) $\vee$ loyalto(Marcus,Caeser)   2

7   loyalto(Marcus,Caeser)

Marcus/X4, Caeser/Y1

1    $\neg$man(Marcus) $\vee$ $\neg$ruler(Caeser) $\vee$ $\neg$tryassassinate(Marcus,Caeser)



[7] Consider the following sentences

- John likes all kinds of food
- Apples are food
- Chicken is food
- Anything anyone eats and isn't killed by is food
- Bill eats peanuts and is still alive
- Sue eats everything Bill eats

(a) Translate these sentences into formulas in predicate logic
(b) Prove that John likes peanuts using **BACKWARD CHAINING**
(c) Convert the formulas of part (a) into clause form
(d) Prove that John likes peanuts using resolution
(e) Use resolution to answer the question "What food does Sue eat?"

a) Translate these sentences into formulas in predicate logic.
The answer:

1. $\forall x$: food(x) → likes(John,x)
2. food (apple)
3. food(Chicken)
4. $\forall x$: $\forall y$: eat(x,y) ∧ ¬killed(x,y) → food(y)
5. eat(Bill,peanuts) ∧ ¬killed(Bill,peanuts)
6. $\forall x$:food(X) ∧ eats(Bill,x) → eats(Sue,x)

b) *Prove that John likes peanuts using backward chaining*

likes(john,peanuts)

↑ *1,substitute*

food(peanuts)

↑ *4,substitute*

$eat(x,peanuts) \wedge \neg killed(x,peanuts)$

↑ 6

$food(peanuts) \wedge eat(Bill,x) \wedge \neg killed(Bill,peanuts)$

↑ *5,substitution*

food(peanuts)

c) *Convert the formulas of part a into clause form*

1. $food(x,y) \rightarrow like(john, x)$
2. $food(apple)$
3. $food\ (chicken)$
4. $\neg eats(x,y) \vee killedby(x,y) \vee food(x,y)$
5. $eats(Bill,peanuts)$
6. $\neg\ killed(Bill,peanuts)$
7. $\neg food(x) \vee \neg eats(Bill,x) \vee eats(Sue,x)$

d) *Prove that John likes peanuts using resolution*

$\neg food(x) \vee like(john,peanuts)$ $\qquad \neg like(john,peanuts)$

peanuts / x

$\neg eat(x,y) \vee killedby(x,y) \vee food(x)$ $\qquad \neg food(x)$

peanuts/y

eats(Bill,peanuts) $\quad$ *¬eat(x,y)∨ killedby(x,y)*

Bill/x

killedby(x,y) $\quad$ *¬ killedby(x,y)*

$\square$

e) *Use resolution to answer the quest, "What food does Sue eat?"*

7 $\quad$ *¬eats(sue,x)∨ eat(Sue,x)*

eats(Bill,peanuts) $\quad$ *¬food(x)∨¬eats(bill,x)∨ eat(Sue,x)*

peanuts / x

4 $\quad$ *¬food(peanuts)∨¬eats(bill,peanuts)*

peanuts/y

¬eats(x,peanuts)∨ killed(x,peanuts)∨ eats(sue,peanuts) $\quad$ *killed(bill,peanuts)*

Bill / x

eats(Bill,peanuts) $\quad$ *¬eats(Bill,peanuts)∨ eats(sue,peanuts)*

eats(sue,peanuts)

[9] Assume the following facts :

- Steve only likes easy courses.
- Science courses are hard.
- All the courses in the basketweaving department are easy.
- BK301 is a basketweaving course.

Use resolution to answer the question, "What course would Steve like?"

The first is to translate these facts into predicates:

1. $\forall x$: easycourse(x) $\rightarrow$ likes(Steve,x)
2. $\neg$ easycourse(science)
3. $\forall x$: basketweaving(x) $\rightarrow$ easycourse(x)
4. basketweaving(Bk301)

The second thing is to put them in the clausal form:

1. $\neg$easycourses(x) $\vee$ likes(Steve,x)
2. $\neg$easycourses(science)
3. $\neg$basketweaving(x) $\vee$ easycourse(x)
4. basketweaving(BK301)

The third is to use the resolution to answer "What courses steve like?"

1 $\qquad$ $\neg$like(steve,x) $\vee$ like(steve,x)

$\neg$ easycourse(x) $\vee$ like(steve,x) $\qquad$ 3

4 $\qquad$ $\neg$basketweaving(x) $\vee$ (steve,x)

bk301 / x

like(steve, bk301)

[10] Consider the following knowledgebase :

$$\forall x : \forall y : cat(x) \wedge fish(y) \rightarrow likes-to-eat(x, y)$$
$$\forall x : calico(x) \rightarrow cat(x)$$
$$\forall x : tuna(x) \rightarrow fish(x)$$

tuna(charlie)
tuna(herb)
calico(puss)

(a) Convert these wffs into **Horn clauses**.

(b) Convert the Horn clauses into a **PROLOG program**.

(c) Write a PROLOG query corresponding to the question, "What does Puss like to eat?" and show how it will be answered by your program.

a) convert there wffs into Horn Clauses.

1)- $\neg cat(x) \vee \neg fish(y) \vee likes\_to\_eat(x,y)$

2)- $\neg calico(x) \vee \neg cat(x)$

3)- $\neg tuma(x) \vee \neg fish(x)$

4)- tuma(charlie)

5)- tuma(herb)

6)- calico(puss)

b)convert the horn clauses into a prolog program

```
1)- likes_to_eat(X,Y):-
        cat(X),fish(Y).
2)- cat(X):-
        calico(X).
3)- fish(X):-
        tuma(X).
4)- tuma(charlie)
5)- tuma(herb)
6)- calico(puss)
```

c)write a prolog query corresponding to the question ,"What does Puss like to eat?" and show how it will be answered by your program.

```
Predicates
        likes_to_eat(symbol,symbol)
        cat(symbol)
        fish(symbol)
        tuma(symbol)
        calico(symbol)
clauses
        likes_to_eat(X,Y):-
                cat(X),fish(Y).
```

```
    fish(X):-
            tuma(charlie)
    tuma(charlie)
    tuma(herb)
    calico(puss)
goal:
    likes_to_eat(puss,What)
output:
    likes_to_eat(puss,charlie)
    likes_to_eat(puss,herb)
    2 Solution
```