9

(2) Convert first order logic to CNF conversion:

Algorithm:-

Step1: Create a list of SKOLEM CONSTANTS. → A, B, C, D

Step2: find ∀, ∃

- If the attributes are lower case, replace them with a skolem constant.
- remove use skolem constant or function from the list
- If the attributes are both lowercase and uppercase replace the uppercase attribute with a skolem function

Step3: replace ⟺ with '→' $P \Leftrightarrow Q$

transform → as $Q \equiv (P \Rightarrow Q) \wedge (Q \Rightarrow P)$

Step4: replace ⇒ with '→' $P \Rightarrow Q$

transform → as $Q \equiv (\sim P) \vee Q$

9 Convert FOL to CNF

(Q) Code

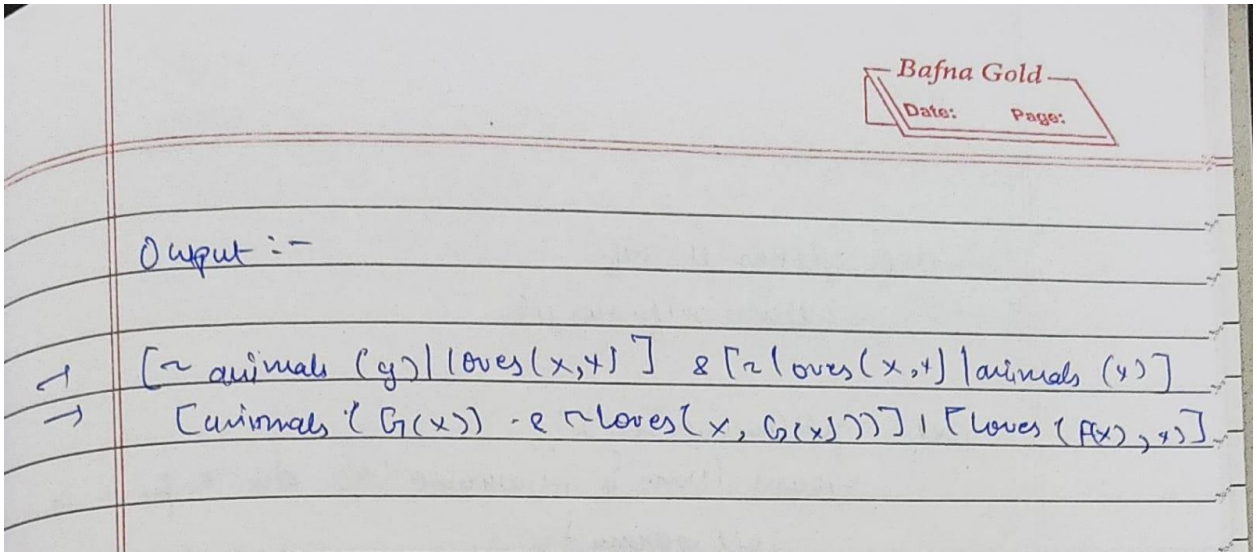
```
def getAttributes(string):
    expr = '\([^)]+\)'
    matches = re.findall(expr, string)
    return [m for m in str(matches) if m.isalpha()]

def getPredicate(string)
    expr : '[a-z~]+\([A-Za-z,]+\)'
    return re.findall(expr, string)

def DeMorgan(sentence)
    string = ''.join(list(sentence).copy())
    string = string.replace('~~','')
    flag = '[' in string

    for predicate in getPredicate(string)
        string = string.replace(predicate, f'~{predicate}')

    s = list(string)

    for i, c in enumerate(string):
        if c=='|':
            s[i] = '&'
        elif c =='&'
            s[i] = '|'

    return f'[{string}]' if flag else string
```

Bafna Gold
Date: Page:

```
def Skolemization(sentence):
    statement = ''.join(list(sentence).copy())
    matches = re.findall('[A-Z].', statement)

    for match in matches[::-1]:
        statement = statement.replace(match, "")

    for s in statements:
        statement = statement.replace(s, s[1:-1])

    for predicate in getPredicates(statements):
        attributes = getAttributes(predicates)
        if ''.join(attributes).islower():
            statement = statement.replace(match[1], pop(0))
        else:
            aL = [a for a in attributes if a.islower()]
            aU = [a for a in attributes if a not a.lower()][0]
```

```
import re

def fol_to_cnf(fol):

    Statement = fol.replace("<=>", "_")

    while '_' in statement:
        i = statement.index('_')
        new = '[' + statement + '=>' + statement
              [i+1:] + ']' & '[' + statement
        + '=>' + statement + ']'
```

```
statement = neg_statement

Statement = statemt.replace("=>", "-")

while '-' in statement:
    i = statement.index('-')
    br = statement.index('(') if '(' in statement else 0

while '~A' in statement:
    i = statement.index('~A')
    statement = list(statement)
    statement[i]; statement[i+1] = '∃'

while '~∃' in statement:
    i = statement.index('~∃')
    s = list(statement)
    statement = ''.join(s)

expr = '(~[A|∃].)'

for i in statements:
    statement = statement.replace(s, DeMorgan(s))
return statement

print(skolemization(fol_to_cnf("animal(y) => loves(x,y)")))
print(skolemization(fol_to_cnf("∀x[∀y[animal(y) => loves
                    => ∃z[loves(z,x)]]")))
```



Output:

In [3]:
```
print(Skolemization(fol_to_cnf("animal(y)<=>loves(x,y)")))
print(Skolemization(fol_to_cnf("∀x[∀y[animal(y)=>loves(x,y)]]=>[∃z[loves(z,x)]]")))
print(fol_to_cnf("[american(x)&weapon(y)&sells(x,y,z)&hostile(z)]=>criminal(x)"))
```

```
[~animal(y)|loves(x,y)]&[~loves(x,y)|animal(y)]
[animal(G(x))&~loves(x,G(x))]|[loves(F(x),x)]
[~american(x)|~weapon(y)|~sells(x,y,z)|~hostile(z)]|criminal(x)
```