Pramod. B.H
1BM18CS070
5-B

A-I LAB TEST-2

2, Given (KB): A => B and C => D query AVC => BVD. use resolution Algo to solve following.

```
def clear():
    global Kb
    Kb = [ ]
def TELL(sentence):
    global Kb
    if isClause(sentence):
        Kb. append(sentence)
    else:
        sentenceCNF = convertCNF(sentence)
        if not sentenceCNF:
            print("Illegal input")
            return
        if isAndList(sentenceCNF):
            for s in sentenceCNF[1:]:
                Kb. append(s)

        else:
            Kb.append(sentenceCNF)
def ASK(sentence):
    global Kb
    if isCLAUSE(sentence):
        neg = negation(sentence)
    else:
        SentenceCNF = convertCNF (sentence)
        if not sentenceCNF:
            print("Illegal input")
            return
        neg = convertCNF(negation(sentenceCNF))
```

Prol(B.y)

```
ask_list=[ ]
if isANDList(neg):
    for n in neg[1:]:
        nCNF= makeCNF(n)
        if type(nCNF).__name__ == 'list':
            ask_list.insert(0, nCNF)
        else:
            ask_list.insert(0, nCNF)

else:
    ask_list= [neg]
    clauses = ask_list + kb[:]

    while True:
        new_clauses = [ ]
        for c1 in clauses:
            for c2 in clauses:
                if c1 is not c2:
                    resolved = resolve(c1,c2)
                    if resolved == False:
                        continue
                    if resolved == [ ]:
                        return True
                    new_clauses.append(resolved)
        if  len(new_clauses) == 0:
            return False

        new_in_clauses = True
        for n in new_clauses:
            if n not in clauses:
                new_in_clauses = False
                clauses.append(n)
```

if new_in_clauses:
return false

Prof [illegible]

return false if

(2)

Bramod B.H
1BM18CS070

```
def resolve(arg-one, arg_two):
    resolved = False
    S1 = make_sentence(arg_one)
    S2 = make_sentence(arg_two)

    resolve_S1 = None
    resolve_S2 = None

    for i in S1:
        if isNotList(i):
            a1 = i[1]
            a1_not = True
        else:
            a1 = i
            a1_not = ~~True~~ False

        for j in S2:
            if isNotList(j):
                a2 = j[1]
                a2_not = True
            else:
                a2 = j
                a2_not = False

            if a1 == a2:
                if a1_not != a2_not:
                    if resolved:
                        return False
                    else:
                        resolved = True
                        resolved_S1 = i
                        resolved_S2 = j
                        break

    if not resolved:
        return False
```

[illegible]

Scanned by CamScanner

```
        s1. remove (resolve_s1)
        s2. remove (resolve_s2)

        result = clear_diplicate (s1+s2)

        if len (result) == 1:
            return result [0]
        elif len (result) > 1:
            result. insert (0, 'or')
        return result
    def make_sentence (arg):
        if isLiteral(arg) or isNotList (arg):
            return [arg]
        if isOrList (arg):
            return clear_duplicate (arg [1:])
        return.
    def clear_duplicate (arg):
        result = [ ]
        for i in rage (0, len(arg)):
            if arg[i] not in arg[i+1:]:
                result. append (arg[i])
        return result.
    def isclause (sentence):
        if isLiteral (sentence):
            return True
        if isNotList (sentence):
            if isLiteral (sentence [1]):
                return True
        else:
            return False.
```

Prod. B.Y

1BM18CS070

```
def convertCNF(sentence):
    while not isCNF(sentence):
        if sentence is None:
            return None
        sentene = makeCNF(sentence)
    return sentence.
def makeCNF(sentence):
    if is Literal(sentence):
        return sentence.
    if (type(sentence).__name__ == 'list'):
        operand = sentence[0]
        if isNotList(sentence):
            if isLiteral(sentence[1]):
                return sentence
            CNF = makeCNF(sentence[1])
            if cnF[0] == 'not':
                return makeCNF(cnF[1])
            if cnF[0] == 'or':
                result = ['and']
                for i in range(1, len(cnF)):
                    result.append(makeCNF(['not', cnF[i]]))
                ~~return~~ result
            if cnF[0] == 'and':
                result = ['or']
                for i in range(1, len(CNF)):
                    result.append(makeCNF(['not', cnf[i]]))
                return result
            return 'False: not'
        if operand == 'implies' and len(sentence) == 3:                     ], makeCNF(sen[2
            return makeCNF(['or', ['not', makeCNF(sentence[1])
```

5

Prod.B.y