```
def negate(term):
    return f'~{term}' if term[0] != '~' else term[1].

def reverse(clause):
    if len(clause) > 2:
        t = split_terms(clause)
        return f'{t[1]} v {t[0]}'
    return.

def split_terms(rule)
    exp = '(~*[ABCD])'
    terms = re.findall(exp, rule)
    return terms.

def contradiction(goal, clause):
    contradiction = [f'{goal} v {negate(goal)}',
                     f'{negate(goal)} v {goal}']
    return clause in contradictions or reverse(clause)
        in contradictions.

def resolve(rules, goal):
    temp = rules.copy()
    temp += [negate(goal)]
    steps = dict()
    for rule in temp:
        steps[rule] = 'Given'.
    steps[negate(goal)] = 'Negated conclusion'
    i = 0
    while i < len(temp):
        n = len(temp)
        j = (i+1) % n.
        clause = []
        while j != i:
```

Nikhil. A.S
1BM18CS061
01/01/2021

```
terms1 = split_terms(temp[i])
terms2 = split_terms(temp[j])
for c in ~~temp~~terms2:
    if negate(c) in term2:
        t1 = [t for t in terms1 if t != c]
        t2 = [t for t in terms2 if t != negate(c)]
        gen = t1 + t2
        if len(gen) == 2:
            if gen[0] != negate(gen[1]):
                clauses += [f'{gen[0]} v {gen[1]}']
            else:
                if contradictional(goal, f'{gen[0]} v {gen[1]}':
                    temp.append(f'{gen[0]} v {gen[1]}')
                    return steps

        else:
            if contradiction(goal, f'{terms1[0]} v {terms2[1]}'):
                temp.append(f'{terms1[0]} v {term2[0]}')
                return steps

for clause in clauses:
    if clause not in temp and clause != reverse(clause)
    not in temp:
        temp.append(clause)
        steps[clause] = f'Resolved from {temp[i]} and {temp[j]}'
j = (j+1) % n

i += 1

return steps.
```

```
def main():
    print("Enter the KB:")
    KB = input()
    Print("Enter the query:")
    query = input()
    KB = KB.split('.')
    steps = resolve(KB, query)
    Print('in step \t clause \t Derivation \t')
    Print('_' * 30)
    i = 1
    for step in steps:
        Print(f'{i}. \t {step} \t {steps[step]} \t')
        i += 1
```