WEEK 6

Create a knowledgebase using prepositional logic and show that the given query entails the knowledge base or not

```
def evaluate_expression(q, p, r):

    # Evaluate the given expression (~q v ~p v r) ^ (~q ^ p) ^ q

    expression_result = ((not q or not p or r) and (not q and p) and q)

    return expression_result


def generate_truth_table():

    # Print the header of the truth table

    print(" q | p | r | Expression (KB) | Query (r)")

    print("---|---|---|-----------------|------------")


    # Evaluate and print each row of the truth table

    for q in [True, False]:

        for p in [True, False]:

            for r in [True, False]:

                expression_result = evaluate_expression(q, p, r)

                query_result = r


                print(f" {q} | {p} | {r} | {expression_result} | {query_result}")


def query_entails_knowledge():

    # Check if query entails the knowledge

    for q in [True, False]:

        for p in [True, False]:

            for r in [True, False]:

                expression_result = evaluate_expression(q, p, r)

                query_result = r
```

```python
        # If the expression is true and the query is false, query does not entail the knowledge
        if expression_result and not query_result:
            return False

    # If the loop completes without returning, query entails the knowledge
    return True

def main():
    # Generate and print the truth table
    generate_truth_table()

    # Check if query entails the knowledge and print the result
    if query_entails_knowledge():
        print("\nQuery entails the knowledge.")
    else:
        print("\nQuery does not entail the knowledge.")

if __name__ == "__main__":
    main()
```

```
 q | p | r | Expression (KB) | Query (r)
---|---|---|-----------------|------------
 True | True | True | False | True
 True | True | False | False | False
 True | False | True | False | True
 True | False | False | False | False
 False | True | True | False | True
 False | True | False | False | False
 False | False | True | False | True
 False | False | False | False | False

Query entails the knowledge.
```

```
 q | p | r | Expression (KB) | Query (r)
---|---|---|-----------------|------------
 True | True | True | True | True
 True | True | False | True | False
 True | False | True | False | False
 True | False | False | True | False
 False | True | True | True | True
 False | True | False | True | False
 False | False | True | False | False
 False | False | False | False | False

Query does not entail the knowledge.
```

WEEK 7

Create a knowledgebase using prepositional logic and prove the given query using resolution.

```
import re


def main(rules, goal):
    rules = rules.split(' ')
    steps = resolve(rules, goal)
    print('\nStep\t|Clause\t|Derivation\t')
    print('-' * 30)
    i = 1
    for step in steps:
        print(f' {i}.\t| {step}\t| {steps[step]}\t')
        i += 1


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

def contradiction(goal, clause):
    contradictions = [ 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
        clauses = []
        while j != i:
            terms1 = split_terms(temp[i])
            terms2 = split_terms(temp[j])
            for c in terms1:
                if negate(c) in terms2:
                    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 contradiction(goal,f'{gen[0]}v{gen[1]}'):
                                temp.append(f'{gen[0]}v{gen[1]}')
                                steps[''] = f"Resolved {temp[i]} and {temp[j]} to {temp[-1]}, which is in turn null. \
                                \nA contradiction is found when {negate(goal)} is assumed as true. Hence, {goal} is true."
                                return steps
                    elif len(gen) == 1:
                        clauses += [f'{gen[0]}']
                    else:
                        if contradiction(goal,f'{terms1[0]}v{terms2[0]}'):
                            temp.append(f'{terms1[0]}v{terms2[0]}')
                            steps[''] = f"Resolved {temp[i]} and {temp[j]} to {temp[-1]}, which is in turn null. \
                            \nA contradiction is found when {negate(goal)} is assumed as true. Hence, {goal} is true."
                            return steps
            for clause in clauses:
                if clause not in temp and clause != reverse(clause) and 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
```

```
[6] rules = 'Rv~P Rv~Q ~RvP ~RvQ' #(P^Q)<=>R : (Rv~P)v(Rv~Q)^(~RvP)^(~RvQ)
    goal = 'R'
    main(rules, goal)
```

```
Step     |Clause  |Derivation
------------------------------
 1.      | Rv~P   | Given.
 2.      | Rv~Q   | Given.
 3.      | ~RvP   | Given.
 4.      | ~RvQ   | Given.
 5.      | ~R     | Negated conclusion.
 6.      |        | Resolved Rv~P and ~RvP to Rv~R, which is in turn null.
A contradiction is found when ~R is assumed as true. Hence, R is true.
```

```
[8] rules = 'PvQ PvR ~PvR RvS Rv~Q ~Sv~Q' # (P=>Q)=>Q, (P=>P)=>R, (R=>S)=>~(S=>Q)
    main(rules, 'R')
```

```
Step     |Clause |Derivation
------------------------------
 1.      | PvQ   | Given.
 2.      | PvR   | Given.
 3.      | ~PvR  | Given.
 4.      | RvS   | Given.
 5.      | Rv~Q  | Given.
 6.      | ~Sv~Q | Given.
 7.      | ~R    | Negated conclusion.
 8.      | QvR   | Resolved from PvQ and ~PvR.
 9.      | Pv~S  | Resolved from PvQ and ~Sv~Q.
 10.     | P     | Resolved from PvR and ~R.
 11.     | ~P    | Resolved from ~PvR and ~R.
 12.     | Rv~S  | Resolved from ~PvR and Pv~S.
 13.     | R     | Resolved from ~PvR and P.
 14.     | S     | Resolved from RvS and ~R.
 15.     | ~Q    | Resolved from Rv~Q and ~R.
 16.     | Q     | Resolved from ~R and QvR.
 17.     | ~S    | Resolved from ~R and Rv~S.
 18.     |       | Resolved ~R and R to ~RvR, which is in turn null.
A contradiction is found when ~R is assumed as true. Hence, R is true.
```