

# Create a knowledge base consisting of first order logic statements and prove the given query using Resolution

```
# Grounded version for John likes peanuts
from copy import deepcopy

def resolve_pair(ci, cj):
    resolvents = []
    for lit_i in ci:
        for lit_j in cj:
            if lit_i.startswith('¬') and lit_i[1:] == lit_j:
                new_clause = list(set(ci + cj))
                new_clause.remove(lit_i)
                new_clause.remove(lit_j)
                resolvents.append(new_clause)
            elif lit_j.startswith('¬') and lit_j[1:] == lit_i:
                new_clause = list(set(ci + cj))
                new_clause.remove(lit_j)
                new_clause.remove(lit_i)
                resolvents.append(new_clause)
    return resolvents

def resolution(KB, query):
    clauses = deepcopy(KB)
    clauses.append(['¬' + query])    # add negated query

    print("Initial Clauses:")
    for c in clauses:
        print(c)

    new = set()
    while True:
        n = len(clauses)
        pairs = [(clauses[i], clauses[j]) for i in range(n) for j in
range(i + 1, n)]
        for (ci, cj) in pairs:
            resolvents = resolve_pair(ci, cj)
            for res in resolvents:
                if not res:
                    print(f"\nResolved {ci} and {cj} -> []")
                    print("☑ Empty clause derived ⇒ Query PROVED!")
                    return True
                new.add(tuple(sorted(res)))

    new_clauses = [list(x) for x in new if list(x) not in clauses]
```

```

        if not new_clauses:
            print("\nNo new clauses ⇒ Query cannot be proved.")
            return False
        for c in new_clauses:
            clauses.append(c)

# --- Grounded Knowledge Base ---
KB = [
    ['¬Food(Apple)', 'Likes(John,Apple)'],          # John likes
    all kinds of food (example instance)
    ['¬Food(Vegetable)', 'Likes(John,Vegetable)'],
    ['¬Food(Peanut)', 'Likes(John,Peanut)'],          # all foods
    liked by John
    ['Food(Apple)'],                                  # Apple is
    food
    ['Food(Vegetable)'],                              # Vegetable
    is food
    ['Alive(Anil)'],                                  # Anil alive
    ['¬Alive(Anil)', 'NotKilled(Anil)'],              # Alive ⇒
    NotKilled
    ['¬NotKilled(Anil)', 'Alive(Anil)'],              # NotKilled ⇒
    Alive
    ['Eats(Anil,Peanut)'],                            # Anil eats
    peanuts
    ['¬Eats(Anil,Peanut)', '¬NotKilled(Anil)', 'Food(Peanut)'], # Eats
    & NotKilled ⇒ Food
]

query = 'Likes(John,Peanut)'

# --- Run resolution ---
resolution(KB, query)

```

OUTPUT:

```

Initial Clauses:
['¬Food(Apple)', 'Likes(John,Apple)']
['¬Food(Vegetable)', 'Likes(John,Vegetable)']
['¬Food(Peanut)', 'Likes(John,Peanut)']
['Food(Apple)']
['Food(Vegetable)']
['Alive(Anil)']
['¬Alive(Anil)', 'NotKilled(Anil)']
['¬NotKilled(Anil)', 'Alive(Anil)']
['Eats(Anil,Peanut)']
['¬Eats(Anil,Peanut)', '¬NotKilled(Anil)', 'Food(Peanut)']
['¬Likes(John,Peanut)']

Resolved ['Alive(Anil)'] and ['Alive(Anil)', '¬Alive(Anil)'] -> []
✅ Empty clause derived ⇒ Query PROVED!
True

```