```python
class Clause:
    def __init__(self, literals):
        self.literals = set(literals)

    def __str__(self):
        return " ∨ ".join(sorted(self.literals))

    def __eq__(self, other):
        return self.literals == other.literals

    def __hash__(self):
        return hash(frozenset(self.literals))

def negate(literal):
    if literal.startswith('¬'):
        return literal[1:]  # Remove negation
    else:
        return '¬' + literal  # Add negation

def resolve(clause1, clause2):
    resolvents = []
    for literal in clause1.literals:
        if negate(literal) in clause2.literals:
            new_clause = (clause1.literals | clause2.literals) - {literal, negate(literal)}
            resolvents.append(Clause(new_clause))
    return resolvents

def resolution(clauses, query):
    # Negate the query
    negated_query = negate(query)
    clauses.append(Clause([negated_query]))

    new_clauses = set(clauses)
    while True:
        pairs = [(c1, c2) for i, c1 in enumerate(new_clauses) for c2 in list(new_clauses)[i + 1:]]
        new_resolvents = set()
        for c1, c2 in pairs:
            resolvents = resolve(c1, c2)
            new_resolvents.update(resolvents)

        # If we derive an empty clause, the original set entails the query
        if Clause(set()) in new_resolvents:
            return True

        # Add new resolvents to the set of clauses
        new_clauses.update(new_resolvents)

        # If no new clauses are generated, we stop
        if new_resolvents.issubset(new_clauses):
            break

    return False

# Example usage
if __name__ == "__main__":
    # Define some clauses
    clauses = [
        Clause(['A', 'B']),
        Clause(['¬A', 'C']),
        Clause(['¬B', '¬C']),
    ]

    # Query to resolve
    query = 'C'

    # Perform resolution
    result = resolution(clauses, query)
    print(f"The query '{query}' is {'entailed' if result else 'not entailed'} by the clauses.")
```

```
The query 'C' is not entailed by the clauses.
```