(Program-7)

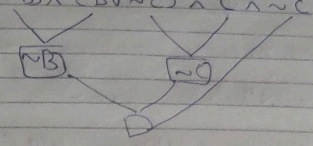
Knowledge Base - Resolution

```
def negate_literal(literal)
    if literal[0] == '~'
        return literal[1:]
    else
        return '~' + literal

def resolve(C1, C2)
    resolved_clause = set(C1) | set(C2)
    for literal in C1:
        if negate_literal(literal) in C2:
            resolved_clause.remove(literal)
                        .remove(negate_literal(literal))
    return tuple(resolved_clause)

def resolution(KB)
    new_clauses = set()
    for i, C1 in enumerate(KB)
        for j, C2 in enumerate(KB)
            if i != j
                new_clause = resolve(C1, C2)
                if len(new_clause) > 0 & new_clause not in
                    new_clauses.add(new_clause)
```

KB: $(A \vee \sim B) \wedge (B \vee \sim C) \wedge C \wedge \sim C$



Algorithm:

1. Main function initializes the resolution process by splitting the input rules and calling 'resolve' function
2. Negate function negates the literal
3. Resolve function performs resolution method it iteratively considers pairs of clauses based on resolution rule until either a contradiction/a goal is reached.

```
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.
```