Name: Rohan Hemant Nemade.
Class: TY IT A
Roll No: 331013
Batch: A1
Title: Implementation of **Unification Algorithm** by considering **Resolution** concept.

Aim:

Assignment 6: Implementation of **Unification Algorithm** by considering **Resolution** concept.

Theory:

1. ***Unification Algorithm***

   - Unification is the process of making two logical expressions identical by finding a suitable substitution of variables.
   - Used in **Predicate Logic, Automated Theorem Proving, and Logic Programming (Prolong).**
   - If two predicates can be made identical with a substitution, they are said to **unify.**

2. ***Resolution Principle***

   - **Definition:** Resolution is a rule of inference used for **propositional and first-order logic**.
   - It is based on **refutation**: proving a statement by showing that its **negation leads to a contradiction**.
   - Widely used in **Automated Reasoning** and **Theorem Proving.**

Algorithm Steps

1. Input two predicates.
2. If they are identical, return success.
3. If one is a variable and the other is a constant/term, substitute.
4. If both are functions with the same name and arity, unify arguments recursively.
5. If mismatch occurs, unification fails.
6. Apply substitutions in Resolution to derive new clauses.
7. If empty clause (⊥) is derived → Contradiction found → Proof complete.

Example and Algorithm:

Step 1: Input

- Take a set of logical expressions (predicates, clauses).
- Define the goal (query) to be proved using resolution.

Step 2: Convert to Clausal Form (CNF)

- Standardize the expressions by:
    1. Eliminating implications (→).
    2. Moving negations inward (¬).
    3. Standardizing variables.
    4. Converting to prenex form and Skolemizing (removing existential quantifiers).
    5. Distributing ∧ over ∨.
    6. Representing the final formula as a set of clauses (disjunctions of literals).

Step 3: Initialize

- Create:
    - Open List (Active Clauses): clauses available for resolution.
    - Substitution Set (θ): stores mappings of variables to terms.

Step 4: Unification Procedure

The unify(p, q) function is used when two literals must match during resolution.

Case 1: If p and q are identical → SUCCESS.
Case 2: If p is a variable → substitute p with q in all clauses (record substitution in θ).
Case 3: If q is a variable → substitute q with p.
Case 4: If both are compound expressions → check operator names and arguments recursively.
Case 5: If mismatch → FAIL (cannot unify).

Step 5: Resolution Rule

1. Select two clauses C1 and C2 that contain complementary literals (e.g., P(x) and ¬P(y)).
2. Apply unification to make them identical.

3. Remove complementary literals and combine remaining literals from both clauses to form a new clause.
4. Add the new clause to the Open List.

Step 6: Check for Empty Clause

- If an empty clause (⊥) is derived → contradiction → proof successful.
- If no new clauses can be generated → resolution fails.

Step 7: Termination

- If the goal is derived (proof complete) → SUCCESS.
- Otherwise, if no progress is possible → FAILURE.

Detailed Example

Knowledge Base (KB):

1. P(x) → Q(x)
2. P(a)
3. Goal: Q(a)

Step 1: Convert to CNF

1. ¬P(x) ∨ Q(x)
2. P(a)
3. Negation of goal: ¬Q(a)

Step 2: Start Resolution

- Clauses: {¬P(x) ∨ Q(x)}, {P(a)}, {¬Q(a)}

Step 3: Apply Resolution

- Resolve {¬P(x) ∨ Q(x)} with {P(a)}
  - Unify P(x) and P(a) → substitution {x → a}
  - New clause: {Q(a)}
- Resolve {Q(a)} with {¬Q(a)}
  - Complementary literals → Resolves to empty clause (⊥)

Step 4: Conclusion

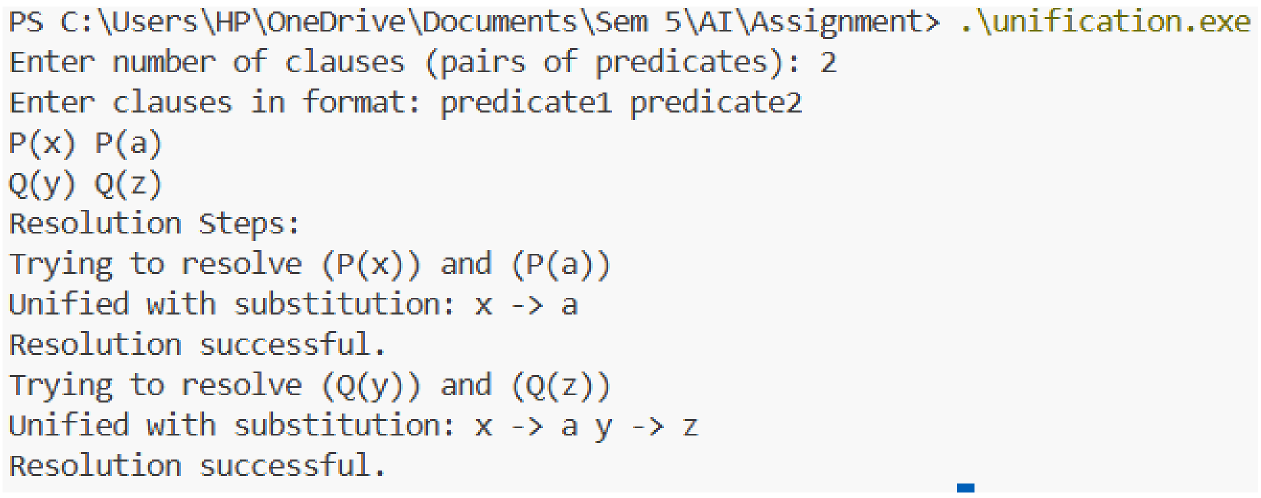
Since ⊥ is derived → Goal Q(a) is proved.

Code:

```
#include <iostream>

#include <vector>

#include <string>

#include <map>

using namespace std;


// Check if a symbol is a variable (lowercase)

bool isVariable(string s) {

   return (s[0] >= 'a' && s[0] <= 'z');

}


map<string, string> substitution;


// Unify arguments

bool unify(string x, string y) {

   if (x == y) return true;
```

```
    if (isVariable(x)) {
        substitution[x] = y;
        return true;
    }
    if (isVariable(y)) {
        substitution[y] = x;
        return true;
    }
    return false; // constants mismatch
}

// Parse predicate into (name, argument)
pair<string,string> parsePredicate(string s) {
    int open = s.find("(");
    int close = s.find(")");
    string pred = s.substr(0, open);          // P
    string arg = s.substr(open + 1, close - open - 1); // x
    return {pred, arg};
}

void resolution(vector<pair<string,string>> clauses) {
    cout << "Resolution Steps:\n";
    for (auto &cl : clauses) {
        auto p1 = parsePredicate(cl.first);
        auto p2 = parsePredicate(cl.second);

        cout << "Trying to resolve (" << cl.first << ") and (" << cl.second << ")\n";
        // predicate names must match
        if (p1.first != p2.first) {
            cout << "Resolution failed: Predicate names differ.\n";
            continue;
        }

        if (unify(p1.second, p2.second)) {
            cout << "Unified with substitution: ";
            for (auto &s : substitution) {
                cout << s.first << " -> " << s.second << " ";
            }
            cout << "\nResolution successful.\n";
        } else {
            cout << "Resolution failed for (" << cl.first << ", " << cl.second << ")\n";
        }
    }
}

int main() {
    int n;
    cout << "Enter number of clauses (pairs of predicates): ";
    cin >> n;

    vector<pair<string,string>> clauses;
    cout << "Enter clauses in format: predicate1 predicate2\n";
    for (int i = 0; i < n; i++) {
        string p1, p2;

        cin >> p1 >> p2;

        clauses.push_back({p1, p2});

    }


    resolution(clauses);

    return 0;

}
```

Output:



```
PS C:\Users\HP\OneDrive\Documents\Sem 5\AI\Assignment> .\unification.exe
Enter number of clauses (pairs of predicates): 2
Enter clauses in format: predicate1 predicate2
P(x) P(a)
Q(y) Q(z)
Resolution Steps:
Trying to resolve (P(x)) and (P(a))
Unified with substitution: x -> a
Resolution successful.
Trying to resolve (Q(y)) and (Q(z))
Unified with substitution: x -> a y -> z
Resolution successful.
```

Conclusion:

The **Unification algorithm** provides a systematic way to substitute variables in logical expressions to make them identical, and the **Resolution principle** applies these substitutions to derive new clauses in logic inference. Together, they form the foundation of **automated theorem proving** and **logic programming (e.g., Prolog)**. Resolution with unification is a complete proof strategy for first-order predicate logic, ensuring correctness when applied systematically.