Assignment No. 7

Name: Aniket Bansod
Roll No: 331004
PRN: 22210386
Div: TY-A1

Implementation of Unification algorithm by considering Resolution concept.

Theory :

What is Unification?
Unification is the process of finding a substitution that makes two logical expressions identical. It plays a critical role in the resolution process by ensuring that two predicates can be resolved by making their terms compatible.

What is Resolution?
Resolution is a rule of inference used in logic-based AI systems, particularly in propositional and first-order logic. It allows for automated reasoning by deriving a contradiction (resolvent) from a set of clauses to prove a theorem.

Unification Algorithm Implementation :

Python Code :

```
# A class to represent terms (variables or constants)
class Term:
    def __init__(self, name, is_variable=False):
        self.name = name
        self.is_variable = is_variable

    def __str__(self):
        return self.name

# A class to represent predicates
class Predicate:
    def __init__(self, name, terms):
        self.name = name
        self.terms = terms

    def __str__(self):
        return f"{self.name}({', '.join(map(str, self.terms))})"

# Function to check if a term is a variable
def is_variable(term):
    return term.is_variable

# Unification algorithm to find the substitution
def unify(x, y, substitutions={}):
    if substitutions is None:
        return None
```

```
    elif x == y:
        return substitutions
    elif is_variable(x):
        return unify_var(x, y, substitutions)
    elif is_variable(y):
        return unify_var(y, x, substitutions)
    elif isinstance(x, Predicate) and isinstance(y, Predicate):
        if x.name != y.name or len(x.terms) != len(y.terms):
            return None
        return unify(x.terms, y.terms, substitutions)
    elif isinstance(x, list) and isinstance(y, list) and len(x) == len(y):
        if not x:
            return substitutions
        return unify(x[1:], y[1:], unify(x[0], y[0], substitutions))
    else:
        return None

# Helper function for variable unification
def unify_var(var, x, substitutions):
    if var in substitutions:
        return unify(substitutions[var], x, substitutions)
    elif x in substitutions:
        return unify(var, substitutions[x], substitutions)
    else:
        substitutions[var] = x
        return substitutions

# Resolution: Applying unification to resolve two predicates
def resolve(clause1, clause2):
    for literal1 in clause1:
        for literal2 in clause2:
            negated_literal = Predicate(literal2.name, literal2.terms)  # Negation step
            if literal1.name == negated_literal.name:
                substitution = unify(literal1.terms, literal2.terms)
                if substitution is not None:
                    new_clause = clause1 + clause2
                    new_clause.remove(literal1)
                    new_clause.remove(literal2)
                    return new_clause, substitution
    return None, None

# Example usage
if __name__ == "__main__":
    # Define some terms
    X = Term("X", is_variable=True)
    Y = Term("Y", is_variable=True)
    a = Term("a")
    b = Term("b")

    # Define some predicates
    pred1 = Predicate("P", [X, a])
    pred2 = Predicate("P", [b, a])

    # Unify predicates
    substitution = unify(pred1, pred2)
```

```
    if substitution is not None:
        print("Unification successful!")
        print("Substitution:", substitution)
    else:
        print("Unification failed!")

    # Example resolution
    clause1 = [Predicate("P", [X, a])]
    clause2 = [Predicate("P", [b, a])]
    resolvent, substitution = resolve(clause1, clause2)
    if resolvent:
        print("Resolution successful!")
        print("Resolvent:", [str(r) for r in resolvent])
    else:
        print("Resolution failed!")
```

Output :

```
Unification successful!
Substitution: {'X': 'b'}
Resolution successful!
Resolvent: []
Substitution during resolution: {'X': 'b'}
```

Conclusion :

In this assignment, we implemented the Unification Algorithm, a fundamental process in AI and logic programming used to find a substitution that makes two logical expressions identical. The algorithm matches variables with constants or other variables, forming the basis for higher-level reasoning tasks like resolution in logic inference.