**Name:** Arya Jalindar Kadam

**Roll No:** 331028

**Div:** A **Batch:** A2 Sub: Artificial Intelligence

# Assignment No - 7

**Title:** Implementation of Unification algorithm by considering Resolution concept.

Theory: Unification and Resolution Algorithms

## Unification Algorithm

The Unification Algorithm is a fundamental component of logical reasoning, especially in artificial intelligence and automated theorem proving. It aims to make two logical expressions or predicates identical by finding suitable substitutions for their variables. Essentially, unification addresses the question: "How can we replace variables to make these expressions the same?"

To illustrate, consider the following statements:

Statement 1: Likes(y, IceCream) (someone likes ice cream)

Statement 2: Likes(Arya, IceCream) (Arya likes ice cream)

In this scenario, the unification algorithm concludes that y must be Arya for both statements to match. The process of replacing y with Arya is called substitution.

The unification process becomes especially useful in automated reasoning systems, where different variables must be resolved to progress in a proof.

## Resolution Algorithm

The Resolution Algorithm is a technique used to resolve logical contradictions or draw conclusions by eliminating opposing statements in logical expressions. It combines information from contradictory statements and removes their differences, retaining only what is consistent.

For example, let's examine two statements:

Statement 1: Likes(y, IceCream) ∨ ¬HasMoney(y) (someone likes ice cream or doesn't have money)

Statement 2: HasMoney(Arya) ∨ ¬Enjoys(Arya, IceCream) (Arya has money or doesn't enjoy ice cream)

The resolution process looks for contradictions. In this case, HasMoney(y) and ¬HasMoney(y) are contradictory because one asserts that someone has money, while the other states that they do not. By unifying y with Arya, we can resolve this contradiction, resulting in a new statement that does not involve the conflicting idea of having or lacking money.

After resolution, we get a simpler conclusion: Conclusion: Likes(Arya, IceCream) ∨ ¬Enjoys(Arya, IceCream)

**Unification**:

In Clause 1, we have a variable y that can be replaced with Arya to make both clauses match. This is where unification comes in, substituting y with Arya in both clauses.

**Resolution**:

Once y is replaced by Arya, we see that the expressions HasMoney(Arya) and ¬HasMoney(Arya) are contradictory. The resolution algorithm resolves this contradiction, yielding a new statement: Likes(Arya, IceCream) ∨ ¬Enjoys(Arya, IceCream).

Thus, based on the provided information, Arya either likes ice cream, or Arya does not enjoy ice cream. The unification and resolution processes collaborate to simplify logical statements by eliminating contradictions.

**Key Concepts in Unification and Resolution**

First-Order Logic (FOL): Both algorithms operate in the context of first-order logic, where statements can include predicates, constants, variables, and functions.

Completeness: The resolution method is complete for first-order logic, meaning that if a conclusion can be drawn from a set of clauses, the resolution algorithm will eventually find it.

**Source Code:**

```
# Unification function

def unify(x, y, theta={}):

    if theta is None:

        return None

    elif x == y:

        return theta

    elif isinstance(x, str) and x.islower():  # Variable case (lowercase = variable)

        return unify_var(x, y, theta)

    elif isinstance(y, str) and y.islower():  # Variable case

        return unify_var(y, x, theta)

    elif isinstance(x, list) and isinstance(y, list):  # Predicate (list) case

        if len(x) != len(y):

            return None

        return unify(x[1:], y[1:], unify(x[0], y[0], theta))

    else:

        return None
```

```
def unify_var(var, x, theta):
    if var in theta:
        return unify(theta[var], x, theta)
    elif x in theta:
        return unify(var, theta[x], theta)
    elif occurs_check(var, x, theta):  # Prevent recursive substitution
        return None
    else:
        theta[var] = x
        return theta
def occurs_check(var, x, theta):
    if var == x:
        return True
    elif isinstance(x, list):
        return any(occurs_check(var, arg, theta) for arg in x)
    elif x in theta:
        return occurs_check(var, theta[x], theta)
    return False


# Resolution function
def resolution(clause1, clause2):
    for lit1 in clause1:
        for lit2 in clause2:
            if lit1 == negation(lit2):
                theta = unify(lit1, lit2)
                if theta is not None:
                    # Remove complementary literals and unify the rest
                    new_clause = (apply_substitution(clause1, theta) +
                            apply_substitution(clause2, theta))
                new_clause = [lit for lit in new_clause if lit != lit1 and lit != lit2]
                return new_clause
    return None


# Negation function
def negation(literal):
    if literal.startswith("¬"):
        return literal[1:]
    else:
        return "¬" + literal


# Apply substitution to clause
def apply_substitution(clause, theta):
    return [substitute(literal, theta) for literal in clause]
# Substitute variables based on the substitution map (theta)
def substitute(literal, theta):
    if isinstance(literal, str):
        for var in theta:
            if var == literal or literal.startswith(var + '('):
                literal = literal.replace(var, theta[var])
        return literal
    return literal


# Example usage with 'Arya'
clause1 = ["Likes(y, IceCream)", "¬HasMoney(y)"]
clause2 = ["HasMoney(Arya)", "¬Enjoys(Arya, IceCream)"]


# Perform resolution
```

```
resolvent = resolution(clause1, clause2)

# Print output

print("Clause 1:", clause1)

print("Clause 2:", clause2)

print("Resolvent:", resolvent)
```

**output**:

```
Clause 1: ['Likes(y, IceCream)', '¬HasMoney(y)']

Clause 2: ['HasMoney(Arya)', '¬Enjoys(Arya, IceCream)']

Resolvent: ['Likes(Arya, IceCream)', '¬Enjoys(Arya, IceCream)']
```

---