LAB 7

UNIFICATION

CODE:

```
#Unification in First Order Logic
def unify(x1, x2):
    """
    Unify two expressions (x1 and x2) based on the given unification
algorithm.
    Returns a substitution set (SUBST) or FAILURE if unification is not
possible.
    """
    if is_variable_or_constant(x1) or is_variable_or_constant(x2):
        if x1 == x2:
            return []  # Return NIL (empty substitution set)
        elif is_variable(x1):
            if occurs_check(x1, x2):
                return "FAILURE"
            else:
                return [(x2, x1)]  # Return {x2/x1}
        elif is_variable(x2):
            if occurs_check(x2, x1):
                return "FAILURE"
            else:
                return [(x1, x2)]  # Return {x1/x2}
        else:
            return "FAILURE"

    if not is_same_predicate(x1, x2):
        return "FAILURE"

    if len(x1) != len(x2):
        return "FAILURE"

    subst = []  # Substitution set

    for i in range(len(x1)):
        s = unify(x1[i], x2[i])
        if s == "FAILURE":
            return "FAILURE"
        elif s:
            subst.extend(s)
            apply_substitution(s, x1[i+1:])
```

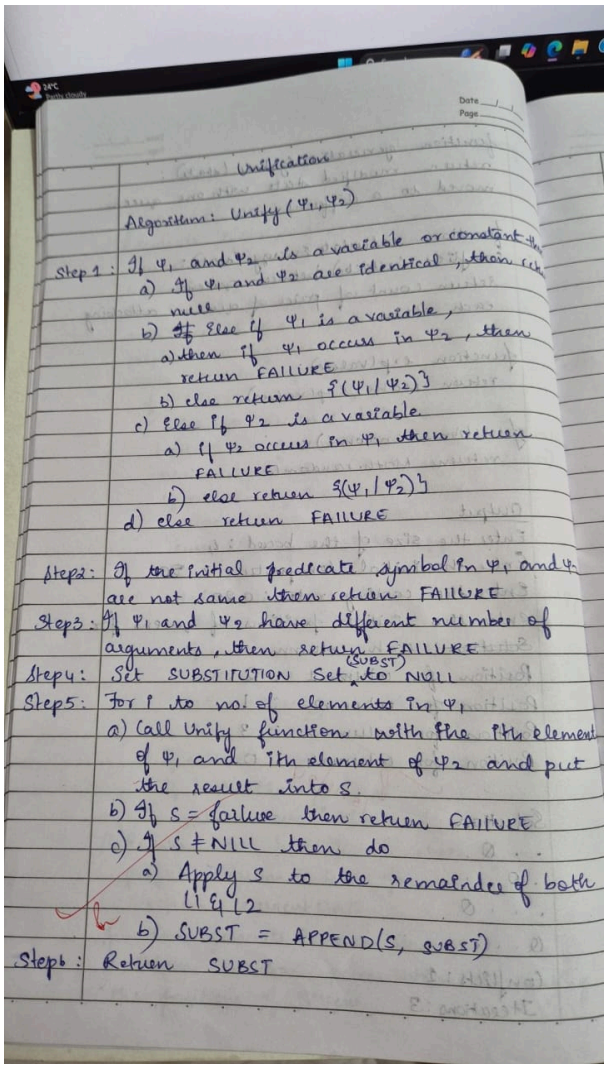
```
            apply_substitution(s, x2[i+1:])

    return subst

def is_variable_or_constant(expr):
    """Check if the expression is a variable or a constant."""
    return isinstance(expr, str) and expr.isalnum()

def is_variable(expr):
    """Check if the expression is a variable."""
    return isinstance(expr, str) and expr.islower()

def occurs_check(var, expr):
    """Check if the variable occurs in the expression."""
    if var == expr:
        return True
    elif isinstance(expr, (list, tuple)):
        return any(occurs_check(var, sub_expr) for sub_expr in expr)
    return False

def is_same_predicate(x1, x2):
    """Check if the initial predicate symbols of x1 and x2 are the same."""
    if isinstance(x1, (list, tuple)) and isinstance(x2, (list, tuple)):
        return x1[0] == x2[0]
    return False

def apply_substitution(subst, expr):
    """Apply the substitution set to the given expression."""
    for old, new in subst:
        if expr == old:
            return new
        elif isinstance(expr, (list, tuple)):
            return [apply_substitution(subst, sub_expr) for sub_expr in expr]
    return expr

def parse_input(expr):
    """Parse user input into a list or tuple representing the predicate."""
    try:
        return eval(expr)
    except Exception as e:
        print(f"Error in input format: {e}")
        return None

# User Input
print("Enter two expressions to unify. Use list/tuple format.")
print("Example: ['P', 'x', 'a'] represents P(x, a)")
```

```
expr1_input = input("Enter the first expression: ")
expr2_input = input("Enter the second expression: ")

expr1 = parse_input(expr1_input)
expr2 = parse_input(expr2_input)

if expr1 is not None and expr2 is not None:
    result = unify(expr1, expr2)
    print("Unification Result:", result)
else:
    print("Invalid input format. Please try again.")

print("Sonal - 1BM22CS286")
```

Output:

```
Enter two expressions to unify. Use list/tuple format.
Example: ['P', 'x', 'a'] represents P(x, a)
Enter the first expression: ['P', 'x', 'a']
Enter the second expression: ['P', 'y', 'a']
Unification Result: [('y', 'x')]
Sonal - 1BM22CS286
```

Observation book Screenshots :



Unification

Algorithm: Unify(Ψ_1, Ψ_2)

Step 1: If Ψ_1 and Ψ_2 is a variable or constant
a) If Ψ_1 and Ψ_2 are identical, then
null
b) Else if Ψ_1 is a variable,
a) then if Ψ_1 occurs in Ψ_2, then
return FAILURE
b) else return $\{(\Psi_1/\Psi_2)\}$
c) Else if Ψ_2 is a variable
a) If Ψ_2 occurs in Ψ_1 then return
FAILURE
b) else return $\{(\Psi_1/\Psi_2)\}$
d) else return FAILURE

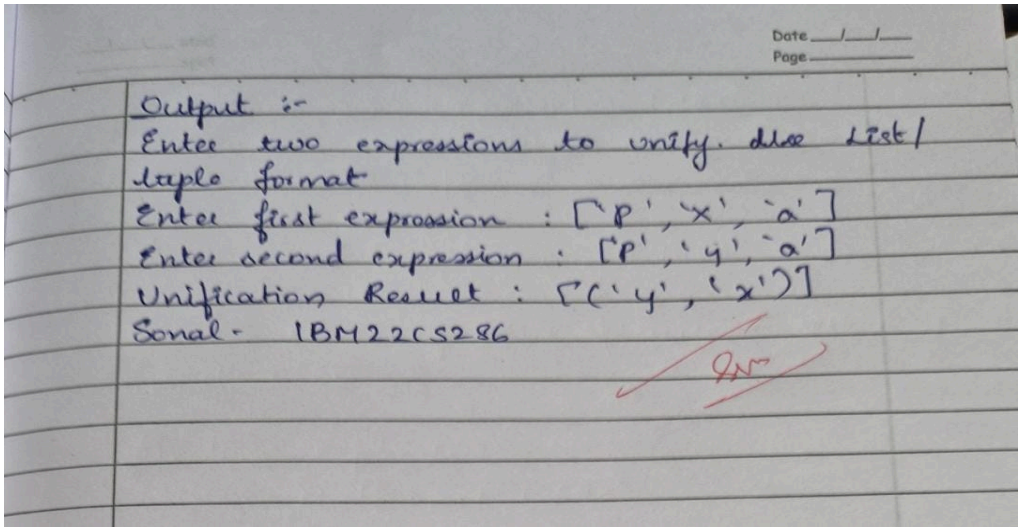
Step2: If the initial predicate symbol in Ψ_1 and Ψ_2 are not same then return FAILURE

Step3: If Ψ_1 and Ψ_2 have different number of arguments, then return FAILURE

Step4: Set SUBSTITUTION Set (SUBST) to NULL

Step5: For i to no. of elements in Ψ_1
a) Call Unify function with the ith element of Ψ_1 and ith element of Ψ_2 and put the result into S.
b) If S = failure then return FAILURE
c) If S ≠ NILL then do
a) Apply S to the remainder of both L1 L2
b) SUBST = APPEND(S, SUBST)

Step6: Return SUBST



Output :-
Enter two expressions to unify. Also list/
tuple format
Enter first expression : ['P', 'x', 'a']
Enter second expression : ['P', 'y', 'a']
Unification Result : [('y', 'x')]
Sonal - 1BM22CS286