22/4/24

LAB-6

Unification Algorithm:

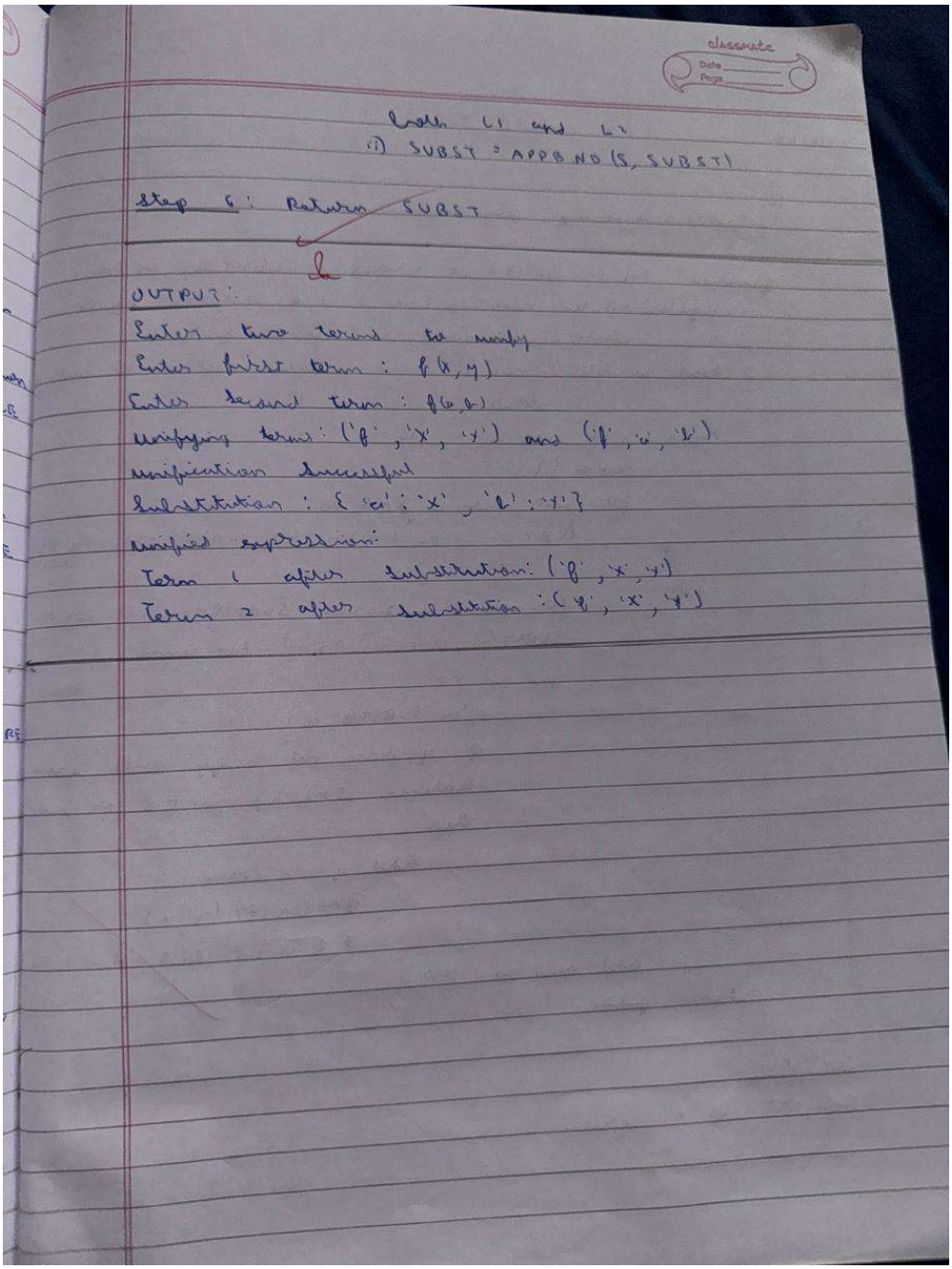
Step 1: If Ψ_1 or Ψ_2 is a variable or constant then
a) If Ψ_1 or Ψ_2 are identical then return NIL.
b) Else if Ψ_1 is a variable,
i) then if Ψ_1 occurs in Ψ_2 then return FAILURE
ii) Else return $\{(\Psi_1 / \Psi_2)\}$
c) Else if Ψ_2 is a variable,
i) If Ψ_2 occurs in Ψ_1 then return FAILURE
ii) Else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

Step 2: If the initial predicate symbol in Ψ_1 and Ψ_2 are not the same then return FAILURE

Step 3: If Ψ_1 and Ψ_2 have a different number of arguments, then return FAILURE

Step 4: Set substitution set (SUBST) to NIL

Step 5: For $i = 1$ to the number of elements in Ψ_1
a) Call Unify function with the i^{th} element of Ψ_1 and i^{th} element of Ψ_2 and put result into S
b) If S = FAILURE then return FAILURE
c) If S ≠ NIL then do,
i) Apply S to the remainder of



Code:

```
def occurs_check(var, term):
    """Check if a variable occurs in a
    term.""" if var == term:
```

```
        return True

    elif isinstance(term, tuple):  # If the term is a function or
        a tuple return any(occurs_check(var, t) for t in
        term[1:])
    return False

def unify(term1, term2,
    substitution=None): """Attempt to unify
    two terms (or predicates)."""
if substitution is None:
        substitution = {}

    # If both terms are the same, no unification
    needed if term1 == term2:
        return substitution

    # If term1 is a variable, try to unify it
    with term2 if isinstance(term1, str) and
    term1.isupper():
        if term1 in substitution:


            return unify(substitution[term1], term2,
        substitution) if occurs_check(term1, term2):
            return None  # Avoid circular unification
        substitution[term1] = term2
        return substitution

    # If term2 is a variable, try to unify it
    with term1 if isinstance(term2, str) and
    term2.isupper():
        return unify(term2, term1, substitution)
```

```
    # If both terms are functions or predicates (tuples), unify their
    components if isinstance(term1, tuple) and isinstance(term2, tuple):
        if len(term1) != len(term2):


            return None  # Different number of
        arguments for t1, t2 in zip(term1[1:],
        term2[1:]):


            substitution = unify(t1, t2,
            substitution) if substitution is
            None:
                return None  # If any unification fails, return
        None return substitution
    return None  # If no other cases match, return None (failure)


# Example usage
term1 = ('P', 'X', 'a')  # Predicate P(X, a)
term2 = ('P', 'b', 'a')  # Predicate P(b, a)

# Attempt to unify
substitution = unify(term1,
term2) if substitution is
not None:
    print("Unification succeeded with substitution:",
substitution) else:
    print("Unification failed")
```

Output:

```
Unification succeeded with substitution: {'X': 'b'}
```