def unify(expr1, expr2, subst=None):

    if subst is None:

        subst = {}

    # Apply substitutions to both expressions

    expr1 = apply\_substitution(expr1, subst)

    expr2 = apply\_substitution(expr2, subst)

    # Base case: Identical expressions

    if expr1 == expr2:

        return subst

    # If expr1 is a variable

    if is\_variable(expr1):

        return unify\_variable(expr1, expr2, subst)

    # If expr2 is a variable

    if is\_variable(expr2):

        return unify\_variable(expr2, expr1, subst)

    # If both are compound expressions (e.g., f(a), P(x, y))

    if is\_compound(expr1) and is\_compound(expr2):

        if expr1[0] != expr2[0] or len(expr1[1]) != len(expr2[1]):

            return None  # Predicate/function symbols or arity mismatch

        for arg1, arg2 in zip(expr1[1], expr2[1]):

            subst = unify(arg1, arg2, subst)

            if subst is None:

                return None

        return subst

    # If they don't unify

    return None

def unify\_variable(var, expr, subst):

    """Handle variable unification."""

    if var in subst:  # Variable already substituted

        return unify(subst[var], expr, subst)

    if occurs\_check(var, expr, subst):  # Occurs-check

        return None

    subst[var] = expr

    return subst

def apply\_substitution(expr, subst):

    """Apply the current substitution set to an expression."""

    if is\_variable(expr) and expr in subst:

        return apply\_substitution(subst[expr], subst)

    if is\_compound(expr):

        return (expr[0], [apply\_substitution(arg, subst) for arg in expr[1]])

    return expr

def occurs\_check(var, expr, subst):

    """Check for circular references."""

    if var == expr:

        return True

    if is\_compound(expr):

        return any(occurs\_check(var, arg, subst) for arg in expr[1])

    if is\_variable(expr) and expr in subst:

        return occurs\_check(var, subst[expr], subst)

    return False

def is\_variable(expr):

    """Check if the expression is a variable."""

    return isinstance(expr, str) and expr.islower()

def is\_compound(expr):

    """Check if the expression is a compound expression."""

    return isinstance(expr, tuple) and len(expr) == 2 and isinstance(expr[1], list)

# Testing the algorithm with the given cases

if \_\_name\_\_ == "\_\_main\_\_":

    # Case 1: p(f(a), f(b)) and p(x, x)

    expr1 = ("p", [("f", ["a"]), ("g", ["b"])])

    expr2 = ("p", ["x", "x"])

    result = unify(expr1, expr2)

    print("Case 1 Result:", result)

    # Case 2: p(b, x, f(g(z))) and p(z, f(y), f(y))

    expr1 = ("p", ["b", "x", ("f", [("g", ["z"])])])

    expr2 = ("p", ["z", ("f", ["y"]), ("f", ["y"])])

    result = unify(expr1, expr2)

    print("Case 2 Result:", result)

