```python
def unify(Y1, Y2, subst=None):
    if subst is None:
        subst = {}

    # Step 1: Check if Y1 or Y2 is a variable or constant
    if Y1 == Y2:  # Identical constants or variables
        print(f"Unification Success: {Y1} and {Y2} are identical.")
        return subst
    elif is_variable(Y1):  # Y1 is a variable
        return unify_variable(Y1, Y2, subst)
    elif is_variable(Y2):  # Y2 is a variable
        return unify_variable(Y2, Y1, subst)

    # Step 2: Predicate symbols not the same
    if predicate_symbol(Y1) != predicate_symbol(Y2):
        print(f"Unification Failure: Predicate symbols {predicate_symbol(Y1)} and {predicate_symbol(Y2)} don't match.")
        return None  # FAILURE

    # Step 3: Different number of arguments
    args1, args2 = arguments(Y1), arguments(Y2)
    if len(args1) != len(args2):
        print(f"Unification Failure: Different number of arguments in {Y1} and {Y2}.")
        return None  # FAILURE

    # Step 5: Recursively unify each element in the lists
    for a1, a2 in zip(args1, args2):
        subst = unify(a1, a2, subst)
        if subst is None:
            return None  # FAILURE

    # Step 6: Return SUBST (final substitution set)
    print(f"Unification Success: {Y1} and {Y2} unified with substitution {subst}.")
    return subst

def unify_variable(var, x, subst):
    if var in subst:
        print(f"Unification Success: Variable {var} is already in the substitution.")
        return unify(subst[var], x, subst)
```

```
    elif occurs_in(var, x):
        print(f"Unification Failure: Variable {var} occurs in {x} (circular reference).")
        return None  # FAILURE due to circular reference
    else:
        print(f"Unification Success: Substituting {var} with {x}.")
        subst[var] = x
        return subst

def predicate_symbol(expr):
    return expr[0] if isinstance(expr, list) else expr

def arguments(expr):
    return expr[1:] if isinstance(expr, list) else []

def is_variable(x):
    return isinstance(x, str) and x.islower()

def occurs_in(var, x):
    if var == x:
        return True
    elif isinstance(x, list):
        return any(occurs_in(var, xi) for xi in x)
    return False

# Example usage: Replace Y1 and Y2 with p(x, f(y)) and p(a, f(g(x)))
Y1 = ['p', 'x', ['f', 'y']]      # p(x, f(y))
Y2 = ['p', 'A', ['f', ['g', 'x']]]  # p(a, f(g(x)))
subst = unify(Y1, Y2)

if subst:
    print("Final Substitution:", subst,"Unification Successful")
else:
    print("Unification failed.")
```

```
Unification Success: Substituting x with A.
Unification Success: Substituting y with ['g', 'x'].
Unification Success: ['f', 'y'] and ['f', ['g', 'x']] unified with substitution {'x': 'A', 'y': ['g', 'x']}.
Unification Success: ['p', 'x', ['f', 'y']] and ['p', 'A', ['f', ['g', 'x']]] unified with substitution {'x': 'A', 'y': ['g', 'x']}.
Final Substitution: {'x': 'A', 'y': ['g', 'x']} Unification Successful
```

```
def unify(Y1, Y2, subst=None):
    if subst is None:
        subst = {}

    # Step 1: Check if Y1 or Y2 is a variable or constant
    if Y1 == Y2:  # Identical constants or variables
        print(f"Unification Success: {Y1} and {Y2} are identical.")
        return subst
    elif is_variable(Y1):  # Y1 is a variable
        return unify_variable(Y1, Y2, subst)
    elif is_variable(Y2):  # Y2 is a variable
        return unify_variable(Y2, Y1, subst)

    # Step 2: Predicate symbols not the same
    if predicate_symbol(Y1) != predicate_symbol(Y2):
        print(f"Unification Failure: Predicate symbols {predicate_symbol(Y1)} and {predicate_symbol(Y2)} don't match.")
        return None  # FAILURE

    # Step 3: Different number of arguments
    args1, args2 = arguments(Y1), arguments(Y2)
    if len(args1) != len(args2):
        print(f"Unification Failure: Different number of arguments in {Y1} and {Y2}.")
        return None  # FAILURE

    # Step 5: Recursively unify each element in the lists
    for a1, a2 in zip(args1, args2):
        subst = unify(a1, a2, subst)
        if subst is None:
            return None  # FAILURE

    # Step 6: Return SUBST (final substitution set)
    print(f"Unification Success: {Y1} and {Y2} unified with substitution {subst}.")
    return subst

def unify_variable(var, x, subst):
    if var in subst:
        print(f"Unification Success: Variable {var} is already in the substitution.")
        return unify(subst[var], x, subst)
    elif occurs_in(var, x):
        print(f"Unification Failure: Variable {var} occurs in {x} (circular reference).")
        return None  # FAILURE due to circular reference
    else:
        print(f"Unification Success: Substituting {var} with {x}.")
        subst[var] = x
        return subst

def predicate_symbol(expr):
    return expr[0] if isinstance(expr, list) else expr

def arguments(expr):
    return expr[1:] if isinstance(expr, list) else []

def is_variable(x):
    return isinstance(x, str) and x.islower()

def occurs_in(var, x):
    if var == x:
        return True
    elif isinstance(x, list):
        return any(occurs_in(var, xi) for xi in x)
    return False

# Example usage: Replace Y1 and Y2 with p(x, f(y)) and p(a, f(g(x)))
Y1 = ['p', 'x', ['f', 'y']]      # p(x, f(y))
Y2 = ['p', 'A', ['f', ['g', 'x']]]  # p(a, f(g(x)))
subst = unify(Y1, Y2)

if subst:
    print("Final Substitution:", subst,"Unification Successful")
else:
    print("Unification failed.")
```

```
Unification Success: Substituting x with A.
Unification Success: Substituting y with ['g', 'x'].
Unification Success: ['f', 'y'] and ['f', ['g', 'x']] unified with substitution {'x': 'A', 'y': ['g', 'x']}.
Unification Success: ['p', 'x', ['f', 'y']] and ['p', 'A', ['f', ['g', 'x']]] unified with substitution {'x': 'A', 'y': ['g', 'x']}
Final Substitution: {'x': 'A', 'y': ['g', 'x']} Unification Successful
```