```
def unify(x, y, subst=None):
    """
    Implements the Unification algorithm for First Order Logic (FOL).
    :param x: First expression
    :param y: Second expression
    :param subst: Current set of substitutions (default: empty dictionary)
    :return: Substitution set if unification is possible, or "FAILURE"
    """
    if subst is None:
        subst = {}

    # Step 1: If x and y are the same, return the current substitutions
    if x == y:
        return subst

    # If x is a variable
    if is_variable(x):
        return unify_var(x, y, subst)

    # If y is a variable
    if is_variable(y):
        return unify_var(y, x, subst)

    # If x and y are compound expressions
    if is_compound(x) and is_compound(y):
        if get_predicate(x) != get_predicate(y):
            return "FAILURE"
        return unify(get_args(x), get_args(y), subst)

    # If x and y are lists
    if isinstance(x, list) and isinstance(y, list):
        if len(x) != len(y):
            return "FAILURE"
        if not x and not y:
            return subst
        return unify(x[1:], y[1:], unify(x[0], y[0], subst))

    # If x and y are constants or cannot be unified
    return "FAILURE"


def unify_var(var, x, subst):
    """
    Handles the unification of a variable with another term.
    :param var: Variable
    :param x: Term to unify with
    :param subst: Current substitution set
    :return: Updated substitution set or "FAILURE"
    """
    if var in subst:
        return unify(subst[var], x, subst)
    if x in subst:
        return unify(var, subst[x], subst)
    if occurs_check(var, x):
        return "FAILURE"
    subst[var] = x
    return subst


def occurs_check(var, x):
    """
    Checks if the variable appears in the term, to prevent infinite loops.
    :param var: Variable
    :param x: Term to check against
    :return: True if var occurs in x, False otherwise
    """
    if var == x:
        return True
    if isinstance(x, list):
        return any(occurs_check(var, arg) for arg in x)
    return False


def is_variable(x):
    """
    Determines if a term is a variable.
    :param x: Term
    :return: True if x is a variable, False otherwise
    """
    return isinstance(x, str) and x.islower()  # Variables are lowercase strings


def is_compound(x):
    """
    Checks if a term is a compound expression.
    :param x: Term
    :return: True if x is compound, False otherwise
    """
    return isinstance(x, str) and '(' in x and ')' in x


def get_predicate(x):
    """
    Extracts the predicate of a compound expression.
    :param x: Compound expression
    :return: Predicate
    """
    return x.split('(')[0]


def get_args(x):
    """
    Extracts the arguments of a compound expression.
    :param x: Compound expression
    :return: List of arguments
    """
    return x[x.index('(') + 1:x.index(')')].split(',')


# Test cases
x1 = "f(x, y)"
x2 = "f(a, b)"
print(unify(x1, x2))  # Expected: {'x': 'a', 'y': 'b'}

x3 = "p(x, g(y))"
x4 = "p(f(a), g(b))"
print(unify(x3, x4))  # Expected: {'x': 'f(a)', 'y': 'b'}
```

```
{'f(x, y)': 'f(a, b)'}
{'p(x, g(y))': 'p(f(a), g(b))'}
```

```
expression_a = "Eats(x, Apple)"
expression_b = "Eats(Riya, y)"

# Perform unification
substitution = unify(expression_a, expression_b)

# Print the result
if substitution == "FAILURE":
    print("Unification failed.")
else:
    print("Unification successful:")
    print(substitution)
```

```
Unification successful:
{'x': 'Riya', ' y': ' Apple'}
```

Start coding or generate with AI.