LAB 7: FOL_Unification

CODE:

```
import re

def occurs_check(var, x):
    """Checks if var occurs in x (to prevent circular substitutions)."""
    if var == x:
        return True
    elif isinstance(x, list):  # If x is a compound expression (like a function or predicate)
        return any(occurs_check(var, xi) for xi in x)
    return False


def unify_var(var, x, subst):
    """Handles unification of a variable with another term."""
    if var in subst:  # If var is already substituted
        return unify(subst[var], x, subst)
    elif isinstance(x, (list, tuple)) and tuple(x) in subst:  # Handle compound expressions
        return unify(var, subst[tuple(x)], subst)
    elif occurs_check(var, x):  # Check for circular references
        return "FAILURE"
    else:
        # Add the substitution to the set (convert list to tuple for hashability)
        subst[var] = tuple(x) if isinstance(x, list) else x
        return subst


def unify(x, y, subst=None):
    """
    Unifies two expressions x and y and returns the substitution set if they can be unified.
    Returns 'FAILURE' if unification is not possible.
    """
    if subst is None:
        subst = {}  # Initialize an empty substitution set

    # Step 1: Handle cases where x or y is a variable or constant
    if x == y:  # If x and y are identical
        return subst
    elif isinstance(x, str) and x.islower():  # If x is a variable
        return unify_var(x, y, subst)
    elif isinstance(y, str) and y.islower():  # If y is a variable
        return unify_var(y, x, subst)
    elif isinstance(x, list) and isinstance(y, list):  # If x and y are compound expressions
(lists)
        if len(x) != len(y):  # Step 3: Different number of arguments
            return "FAILURE"

        # Step 2: Check if the predicate symbols (the first element) match
        if x[0] != y[0]:  # If the predicates/functions are different
            return "FAILURE"

        # Step 5: Recursively unify each argument
        for xi, yi in zip(x[1:], y[1:]):  # Skip the predicate (first element)
            subst = unify(xi, yi, subst)
            if subst == "FAILURE":
                return "FAILURE"
        return subst
    else:  # If x and y are different constants or non-unifiable structures
```

```python
        return "FAILURE"


def unify_and_check(expr1, expr2):
    """
    Attempts to unify two expressions and returns a tuple:
    (is_unified: bool, substitutions: dict or None)
    """
    result = unify(expr1, expr2)
    if result == "FAILURE":
        return False, None
    return True, result


def display_result(expr1, expr2, is_unified, subst):
    print("Expression 1:", expr1)
    print("Expression 2:", expr2)
    if not is_unified:
        print("Result: Unification Failed")
    else:
        print("Result: Unification Successful")
        print("Substitutions:", {k: list(v) if isinstance(v, tuple) else v for k, v in
subst.items()})


def parse_input(input_str):
    """Parses a string input into a structure that can be processed by the unification
algorithm."""
    # Remove spaces and handle parentheses
    input_str = input_str.replace(" ", "")

    # Handle compound terms (like p(x, f(y)) -> ['p', 'x', ['f', 'y']])
    def parse_term(term):
        # Handle the compound term
        if '(' in term:
            match = re.match(r'([a-zA-Z0-9_]+)\((.*)\)', term)
            if match:
                predicate = match.group(1)
                arguments_str = match.group(2)
                arguments = [parse_term(arg.strip()) for arg in arguments_str.split(',')]
                return [predicate] + arguments
        return term

    return parse_term(input_str)


# Main function to interact with the user
def main():
    while True:
        # Get the first and second terms from the user
        expr1_input = input("Enter the first expression (e.g., p(x, f(y))): ")
        expr2_input = input("Enter the second expression (e.g., p(a, f(z))): ")

        # Parse the input strings into the appropriate structures
        expr1 = parse_input(expr1_input)
        expr2 = parse_input(expr2_input)

        # Perform unification
        is_unified, result = unify_and_check(expr1, expr2)
```

```python
        # Display the results
        display_result(expr1, expr2, is_unified, result)

        # Ask the user if they want to run another test
        another_test = input("Do you want to test another pair of expressions? (yes/no): 
").strip().lower()
        if another_test != 'yes':
            break


if __name__ == "__main__":
    main()
```

RESULT:

```
Enter the first expression (e.g., p(x, f(y))): q(a,g(x,a),f(y))
Enter the second expression (e.g., p(a, f(z))): q(a,g(f(b),a),x)
Expression 1: ['q', 'a', 'g(x', 'a)', ['f', 'y']]
Expression 2: ['q', 'a', ['g', 'f(b'], 'a)', 'x']
Result: Unification Successful
Substitutions: {'g(x': ['g', 'f(b'], 'x': ['f', 'y']}
Do you want to test another pair of expressions? (yes/no): yes
Enter the first expression (e.g., p(x, f(y))): p(z,x,f(g(z))
Enter the second expression (e.g., p(a, f(z))): p(z,f(y),f(y))
Expression 1: ['p', 'z', 'x', ['f', 'g(z']]
Expression 2: ['p', 'z', ['f', 'y'], ['f', 'y']]
Result: Unification Successful
Substitutions: {'x': ['f', 'y'], 'g(z': 'y'}
Do you want to test another pair of expressions? (yes/no): yes
Enter the first expression (e.g., p(x, f(y))): p(f(a),g(x))
Enter the second expression (e.g., p(a, f(z))): p(x,x)
Expression 1: ['p', ['f', 'a'], ['g', 'x']]
Expression 2: ['p', 'x', 'x']
Result: Unification Failed
Do you want to test another pair of expressions? (yes/no): no
```