```
def is_variable(term):
    """Check if a term is a variable."""
    return isinstance(term, str) and term.islower()

def is_constant(term):
    """Check if a term is a constant."""
    return isinstance(term, str) and term.isupper()

def unify(term1, term2, subst=None):
    """
    Unify two terms.
    Args:
        term1: The first term (variable, constant, or function).
        term2: The second term (variable, constant, or function).
        subst: Current set of substitutions (dictionary).
    Returns:
        A substitution dictionary if unification is successful, otherwise None.
    """
    if subst is None:
        subst = {}

    if term1 == term2:  # If terms are identical
        return subst

    if is_variable(term1):  # If term1 is a variable
        return unify_variable(term1, term2, subst)

    if is_variable(term2):  # If term2 is a variable
        return unify_variable(term2, term1, subst)

    if isinstance(term1, tuple) and isinstance(term2, tuple):
        # If terms are functions, unify their name and arguments
        if term1[0] != term2[0] or len(term1[1]) != len(term2[1]):
            return None  # Function names or argument lengths differ
        for arg1, arg2 in zip(term1[1], term2[1]):
            subst = unify(arg1, arg2, subst)
            if subst is None:
                return None
        return subst

    return None  # Terms cannot be unified


def unify_variable(var, term, subst):
    """
    Unify a variable with a term.
    Args:
        var: The variable (string).
        term: The term to unify with (variable, constant, or function).
        subst: Current set of substitutions (dictionary).
    Returns:
        Updated substitution dictionary or None.
    """
    if var in subst:  # Variable already substituted
        return unify(subst[var], term, subst)

    if occurs_check(var, term, subst):  # Prevent infinite loops
        return None

    subst[var] = term
    return subst


def occurs_check(var, term, subst):
    """
    Check if a variable occurs in a term (to prevent infinite loops).
    Args:
        var: The variable (string).
        term: The term to check against.
        subst: Current set of substitutions (dictionary).
    Returns:
        True if var occurs in term, False otherwise.
    """
    if var == term:
        return True
    if isinstance(term, tuple):  # If term is a function, check its arguments
        return any(occurs_check(var, arg, subst) for arg in term[1])
    if var in subst and occurs_check(var, subst[var], subst):
        return True
    return False


def apply_substitution(term, subst):
    """
    Apply a substitution to a term.
    Args:
        term: The term to substitute (variable, constant, or function).
        subst: The substitution dictionary.
    Returns:
        The term after applying the substitution.
    """
    if is_variable(term) and term in subst:
        return apply_substitution(subst[term], subst)
    if isinstance(term, tuple):  # If the term is a function, apply substitutio
        return (term[0], [apply_substitution(arg, subst) for arg in term[1]])
    return term  # Return the term as-is for constants or unbound variables
```

```python
# Example Usage
if __name__ == "__main__":
    # Example terms:
    term1 = ("f", ["x", "y"])  # f(x, y)
    term2 = ("f", ["a", "b"])  # f(a, b)

    # Perform unification
    result = unify(term1, term2)
    if result:
        print("Unification successful! Substitution:")
        print(result)

        # Apply substitution to the original terms
        term1_substituted = apply_substitution(term1, result)
        term2_substituted = apply_substitution(term2, result)

        print("\nTerms after substitution:")
        print(f"Term 1: {term1_substituted}")
        print(f"Term 2: {term2_substituted}")
    else:
        print("Unification failed.")
```

```
Unification successful! Substitution:
{'x': 'a', 'y': 'b'}

Terms after substitution:
Term 1: ('f', ['a', 'b'])
Term 2: ('f', ['a', 'b'])
```