11/1/23

Lab 8 - Proof by Unification:

Aim To implement unification in First order logic

```
import re

def getAttributes(expression):
    expression = expression.split("(")[1:][0]
    expression = "(".join(expression)
    expression = expression.split(")")[:-1]
    expression = ")".join(expression)
    attributes = expression.split(',')
    return attributes

def getInitialPredicate(expression):
    return expression.split("(")[0]

def isConstant(char):
    return char.isupper() and len(char) == 1

def isVariable(char):
    return char.islower() and len(char) == 1

def replaceAttributes(exp, old, new):
    attributes = getAttributes(exp)
    predicate = getInitialPredicate(exp)
    for index, val in enumerate(attributes):
        if val == old:
            attributes[index] = new
    return predicate + "(" + ",".join(attributes) + ")"

def apply(exp, substitutions):
    for substitution in substitutions:
```

```
        new_old = substitution
        exp = replaceAttribute(exp, old, new)
    return exp

def checkOccurs(var, exp):
    if exp.find(var) == -1:
        return False
    return True

def getFirstPart(expression):
    attributes = getAttributes(expression)
    return attributes[0]

def getRemainingPart(expression):
    predicate = getInitialPredicate(expression)
    attributes = getAttributes(expression)
    newExpression = predicate + "(" + ",".join
        (attributes[1:]) + ")"
    return newExpression

def unify(exp1, exp2):
    if exp1 == exp2:
        return []

    if isConstant(exp1) and isConstant(exp2):
        if exp1 != exp2:
            print(f"{exp1} and {exp2} are
                constants. Cannot be unified")
        return []

    if isConstant(exp1):
        return [(exp2, exp1)] if not checkOccurs
                (exp2, exp1) else []
    if isConstant(exp2):
```

classmate
Date
Page

return [(exp2, exp1)].

if isVariable(exp2):
    return [(exp2, exp1)] if not ~~check~~
    checkOccurs(exp2, exp1) else []

if getInitialPredicate(exp1) != getInitialPredicate(exp2):
    print("Cannot be unified as the predicates do not match")
    return [].

attributeCount1 = len(getAttributes(exp1))
attributeCount2 = len(getAttributes(exp2))
if attributeCount1 != AttributeCount2:
    print(f"Length of attributes {attributeCount1} and {attributeCount2} do not match. Cannot be unified")
    return []

head1 = getFirstPart(exp1)
head2 = getFirstPart(exp2).
initialSubstitution = unify(head1, head2)
if not initalSubstitution:
    return [].
if attributeCount1 == 1:
    return initialSubstitution

tail1 = getRemainingPart(exp1)
tail2 = getRemainingPart(exp2)

if initialSubstitution.) = []:
    tail1 = apply(tail1, initialSubstitution)
    tail2 = apply(tail2, initialSubstitution)

```
    remaining_substitution = unify(tail1, tail2)
    print remaining_substitution
    return []

return initial_substitution + remaining_substitution

if __name__ == "__main__":
    print("Enter the first expression")
    e1 = input()
    print("Enter the second expression")
    e2 = input()
    substitutions = unify(e1, e2)
    print("The substitutions are:")
    print([','.join(substitution) for substitution in substitutions]).
```

output 1: Enter the first expression.
knows(f(x),y)
Enter the second expression
knows(J, John)
The substitutions are:
['J / f(x)', 'John / y']

Enter the first expression
like(A,y)
Enter the second expression
like(k, g(k))
A and k are constants. Cannot be unified
The substitutions are:
[].

Enter the first expression
Student (x)
Enter the second expression
Teacher (Ron)
Cannot be unified as the predicates do not match!
The substitutions are:
[].

Output: Enter your first expression
knows (f(x), y)
Enter the second expression
knows (J, John)
The substitutions are:
['J / f(x)', 'John / y']

Enter the first expression
like (A, y)
Enter the second expression
like (K, g(x))
A and K are constants. Cannot be unified.
The substitutions are
[].

Enter the first expression
Student (x)
Enter the second expression
Teacher (Ron)
Cannot be unified as the predicates do not match.
The substitutions are:
[].

Algorithm:

function Unify(x, y, θ) returns a substitution to make x and y identical
inputs: x, a variable, a constant, list, or compound expression
y, a variable, constant, list or compound expression
θ, the substitution built up so far.
(optional, defaults to empty).
if value = failure then return failure.
else if x = y then return θ
else if variable?(x) then return Unify Var(x, y, θ)
else if variable?(y) then return Unify Var(y, x, θ)
else if compound?(x) and compound?(y) then
return Unify(x.args, y.args, Unify(x.op, y.op, θ))
else if LIST?(x) and LIST?(y) then
return Unify(x.REST, y.REST, Unify(x.FIRST, y.FIRST, θ))
else return failure.

function Unify-Var(var, x, θ) returns a substitution

if {var/val} ∈ θ then return Unify(val, x, θ)
else if {x/val} ∈ θ then return Unify(var, val, θ)
else if Occur-Check?(var, x) then return failure
else return add {val/x} to θ