12/1/2024

Unification

Eg: Knows(John, x) Knows(John, Jane)

{x/Jane}

Step1: If term1 or term2 is a variable or constant, then:

a) term1 or term2 are identical

return NIL

b) Else if term1 is a variable

if term1 occurs in term2

return FAIL

else

return {(term2/term1)}

c) else if term2 is a variable

if term2 occurs in term1

return FAIL

else

return {(term1/term2)}

d) else return FAIL

Step2: If predicate(term1) ≠ predicate(term2)

return FAIL

Step3: number of arguments ≠

return FAIL

Step4: set(SUBST) to NIL

Step5: For i=1 to the number of elements in term1

a) Call unify (ith term1, ith term2)

put result into S

b) S= FAIL

return FAIL

c) If $S \neq NIL$

a. Apply S to the remainder of both t_1 and t_2

b. SUBST = APPEND (S, SUBST)

Step 6: Return SUBST

Unification - code:

```
import re

def getAttributes(expression):
    expression = expression.split("(")[1:]
    expression = "(".join(expression)
    expression = expression[:-1]
    expression = re.split("(?<!\()(,)(?!\))", expression)
    return expression

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

def apply(exp, substitutions):
    for substitution in substitutions:
        new, old = substitution
        exp = replaceAttributes(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: return False
    if isConstant(exp1):
        return [(exp1, exp2)]
    if isConstant(exp2):
        return [(exp2, exp1)]
    if isVariable(exp1):
        if checkOccurs(exp1, exp2): return False
        else: return [(exp2, exp1)]
    if isVariable(exp2):
        if checkOccurs(exp2, exp1): return False
        else: return [(exp1, exp2)]
    if getInitialPredicate(exp1) != getInitialPredicate(exp2):
        print("Predicates do not match. Cannot be unified")
        return False

    head1 = getFirstPart(exp1)
    head2 = getFirstPart(exp2)
    initialSubstitution = unify(head1, head2)
    return initialSubstitution
```

→ exp1 = "knows(x)"
exp2 = "knows(Richard)"
Output:
[('x', 'Richard')]

→ exp1 = "knows(A, x, y)"
exp2 = "knows(y, mother(y))"
Output:
Substitutions:
False

→ exp1 = "knows(A, x)"
exp2 = "k(y, mother(y))"
substitution = unify(exp1, exp2)
Output:
Substitutions
False

```
[9]  exp1 = "knows(x)"
     exp2 = "knows(Richard)"
     substitutions = unify(exp1, exp2)
     print("Substitutions:")
     print(substitutions)
```

```
Substitutions:
[('Richard', 'x')]
```

```
[7]  exp1 = "knows(A,x)"
     exp2 = "k(y,mother(y))"
     substitutions = unify(exp1, exp2)
     print("Substitutions:")
     print(substitutions)
```

```
Substitutions:
[('A', 'y'), ('mother(y)', 'x')]
```

```
exp1 = "knows(A,x)"
exp2 = "knows(y)"
substitutions = unify(exp1, exp2)
print("Substitutions:")
print(substitutions)
```