AI EXPERIMENT 8
Implementation of unification and resolution for real world problems.

Team AI4life :-
Sai Mohit Ambekar (137)
Sadekar Adesh H. (141)
Kapuluru Shrinivasulu (142)
Praneet Botke (149)
Aayushi Goenka (151)
Sonia Raja (152)

Problem Statement :

Develop a program to unify expressions and direct the output of resolution to output.txt after taking input from input.txt file in same directory.

What is Unification?

- Unification is a process of making two different logical atomic expressions identical by finding a substitution. Unification depends on the substitution process.
- It takes two literals as input and makes them identical using substitution.
- Let Ψ_1 and Ψ_2 be two atomic sentences and σ be a unifier such that, $\Psi_1\sigma = \Psi_2\sigma$, then it can be expressed as UNIFY(Ψ_1, Ψ_2).
- Example: Find the MGU for Unify{King(x), King(John)}

Let Ψ_1 = King(x), Ψ_2 = King(John),

Substitution θ = {John/x} is a unifier for these atoms and applying this substitution, and both expressions will be identical.

- The UNIFY algorithm is used for unification, which takes two atomic sentences and returns a unifier for those sentences (If any exist).
- Unification is a key component of all first-order inference algorithms.
- It returns fail if the expressions do not match with each other.
- The substitution variables are called Most General Unifier or MGU.

Example:

Let's say there are two different expressions, P(x, y), and P(a, f(z)).

In this example, we need to make both above statements identical to each other. For this, we will perform the substitution.

P(x, y)..... (i)
P(a, f(z))..... (ii)

- Substitute x with a, and y with f(z) in the first expression, and it will be represented as a/x and f(z)/y.
- With both the substitutions, the first expression will be identical to the second expression and the substitution set will be: [a/x, f(z)/y].

Unification Algorithm:

Algorithm: Unify(Ψ_1, Ψ_2)

Step. 1: If Ψ_1 or Ψ_2 is a variable or constant, then:
a) If Ψ_1 or Ψ_2 are identical, then return NIL.
b) Else if $\Psi_1$is a variable,
a. then if Ψ_1 occurs in Ψ_2, then return FAILURE
b. Else return { (Ψ_2/ Ψ_1)}.
c) Else if Ψ_2 is a variable,

a. If Ψ_2 occurs in Ψ_1 then return FAILURE,

b. Else return {(Ψ_1/ Ψ_2)}.

d) Else return FAILURE.

Step.2: If the initial Predicate symbol in Ψ_1 and Ψ_2 are not same, then return FAILURE.

Step. 3: IF Ψ_1 and Ψ_2 have a different number of arguments, then return FAILURE.

Step. 4: Set Substitution set(SUBST) to NIL.

Step. 5: For i=1 to the number of elements in Ψ_1.

a) Call Unify function with the ith element of Ψ_1 and ith element of Ψ_2, and put the result into S.

b) If S = failure then returns Failure

c) If S ≠ NIL then do,

a. Apply S to the remainder of both L1 and L2.

b. SUBST= APPEND(S, SUBST).

Step.6: Return SUBST.

Implementation of Unification Algorithm:

For each pair of the following atomic sentences find the most general unifier (If exist).

1. Find the MGU of {p(f(a), g(Y)) and p(X, X)}

   a. Sol: S_0 => Here, Ψ_1 = p(f(a), g(Y)), and Ψ_2 = p(X, X)

   SUBST θ= {f(a) / X}

   S1 => Ψ_1 = p(f(a), g(Y)), and Ψ_2 = p(f(a), f(a))

   SUBST θ= {f(a) / g(y)}, Unification failed.

   Unification is not possible for these expressions.

2. Find the MGU of {p(b, X, f(g(Z))) and p(Z, f(Y), f(Y))}

   Here, Ψ_1 = p(b, X, f(g(Z))) , and Ψ_2 = p(Z, f(Y), f(Y))

   S_0 => { p(b, X, f(g(Z))); p(Z, f(Y), f(Y))}

   SUBST θ={b/Z}

S_1 => { p(b, X, f(g(b))); p(b, f(Y), f(Y))}

SUBST θ={f(Y) /X}

S_2 => { p(b, f(Y), f(g(b))); p(b, f(Y), f(Y))}

SUBST θ= {g(b) /Y}

S_2 => { p(b, f(g(b)), f(g(b)); p(b, f(g(b)), f(g(b))} Unified Successfully.

And Unifier = { b/Z, f(Y) /X , g(b) /Y}.

3. Find the MGU of {p (X, X), and p (Z, f(Z))}

   Here, Ψ_1 = {p (X, X), and Ψ_2 = p (Z, f(Z))

   S_0 => {p (X, X), p (Z, f(Z))}

   SUBST θ= {X/Z}

   S1 => {p (Z, Z), p (Z, f(Z))}

   SUBST θ= {f(Z) / Z}, Unification Failed.

   Hence, unification is not possible for these expressions.

4. Find the MGU of UNIFY(prime (11), prime(y))

   Here, Ψ_1 = {prime(11) , and Ψ_2 = prime(y)}

   S_0 => {prime(11) , prime(y)}

   SUBST θ= {11/y}

   S_1 => {prime(11) , prime(11)} , Successfully unified.

   Unifier: {11/y}.

5. Find the MGU of Q(a, g(x, a), f(y)), Q(a, g(f(b), a), x)}

   Here, Ψ_1 = Q(a, g(x, a), f(y)), and Ψ_2 = Q(a, g(f(b), a), x)

   S_0 => {Q(a, g(x, a), f(y)); Q(a, g(f(b), a), x)}

   SUBST θ= {f(b)/x}

   S_1 => {Q(a, g(f(b), a), f(y)); Q(a, g(f(b), a), f(b))}

SUBST θ= {b/y}

S_1 => {Q(a, g(f(b), a), f(b)); Q(a, g(f(b), a), f(b))}, Successfully Unified.

Unifier: [a/a, f(b)/x, b/y].

6. UNIFY(knows(Richard, x), knows(Richard, John))

   Here, Ψ_1 = knows(Richard, x), and Ψ_2 = knows(Richard, John)

   S_0 => { knows(Richard, x); knows(Richard, John)}

   SUBST θ= {John/x}

   S_1 => { knows(Richard, John); knows(Richard, John)}, Successfully Unified.

   Unifier: {John/x}.

Code:

```
def get_index_comma(string):

    index_list = list()

    par_count = 0



    for i in range(len(string)):

        if string[i] == ',' and par_count == 0:

            index_list.append(i)

        elif string[i] == '(':

            par_count += 1

        elif string[i] == ')':

            par_count -= 1
```

```
    return index_list



def is_variable(expr):
    for i in expr:
        if i == '(' or i == ')':
            return False

    return True



def process_expression(expr):
    expr = expr.replace(' ', '')
    index = None
    for i in range(len(expr)):
        if expr[i] == '(':
            index = i
            break
    predicate_symbol = expr[:index]
    expr = expr.replace(predicate_symbol, '')
    expr = expr[1:len(expr) - 1]
    arg_list = list()
    indices = get_index_comma(expr)
```

```
    if len(indices) == 0:
        arg_list.append(expr)
    else:
        arg_list.append(expr[:indices[0]])
        for i, j in zip(indices, indices[1:]):
            arg_list.append(expr[i + 1:j])
        arg_list.append(expr[indices[len(indices) - 1] + 1:])

    return predicate_symbol, arg_list


def get_arg_list(expr):
    _, arg_list = process_expression(expr)

    flag = True
    while flag:
        flag = False

        for i in arg_list:
            if not is_variable(i):
                flag = True
                _, tmp = process_expression(i)
                for j in tmp:
                    if j not in arg_list:
```

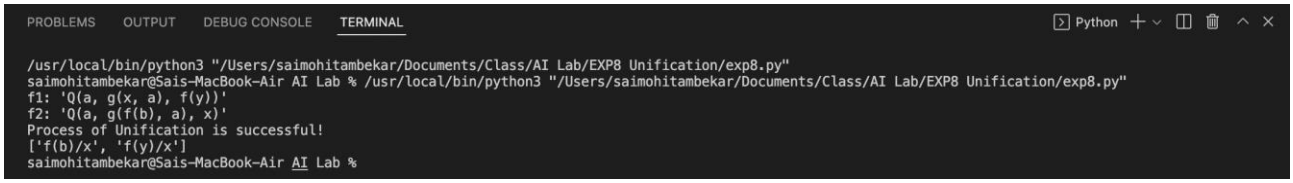
```
                arg_list.append(j)
            arg_list.remove(i)

    return arg_list


def check_occurs(var, expr):
    arg_list = get_arg_list(expr)
    if var in arg_list:
        return True

    return False


def unify(expr1, expr2):

    if is_variable(expr1) and is_variable(expr2):
        if expr1 == expr2:
            return 'Null'
        else:
            return False
    elif is_variable(expr1) and not is_variable(expr2):
        if check_occurs(expr1, expr2):
            return False
```

```
        else:
            tmp = str(expr2) + '/' + str(expr1)
            return tmp
    elif not is_variable(expr1) and is_variable(expr2):
        if check_occurs(expr2, expr1):
            return False
        else:
            tmp = str(expr1) + '/' + str(expr2)
            return tmp
    else:
        predicate_symbol_1, arg_list_1 = process_expression(expr1)
        predicate_symbol_2, arg_list_2 = process_expression(expr2)

        # Step 2
        if predicate_symbol_1 != predicate_symbol_2:
            return False
        # Step 3
        elif len(arg_list_1) != len(arg_list_2):
            return False
        else:
            # Step 4: Create substitution list
            sub_list = list()

            # Step 5:
```

```
        for i in range(len(arg_list_1)):
            tmp = unify(arg_list_1[i], arg_list_2[i])

            if not tmp:
                return False
            elif tmp == 'Null':
                pass
            else:
                if type(tmp) == list:
                    for j in tmp:
                        sub_list.append(j)
                else:
                    sub_list.append(tmp)

        # Step 6
        return sub_list


if _name_ == '_main_':

    f1 = 'Q(a, g(x, a), f(y))'
    f2 = 'Q(a, g(f(b), a), x)'
    # f1 = input('f1 : ')
    # f2 = input('f2 : ')
```

```
result = unify(f1, f2)
if not result:
    print('Process of Unification has failed!')
else:
    print(f"f1: '{f1}'")
    print(f"f2: '{f2}'")
    print('Process of Unification is successful!')
    print(result)
```

Output:



Result:

Unification of expression was done and the conversion set was printed and the result of all queries in input file were printed and written to output.txt