Batch 1 :

PA17 Ketaki Patil

AI Lab Assignment no. 4

Title : Implementation of unification algorithm.

Aim : To implement unification algorithm.

Objective : To study and implement unification algorithm.

Theory :

① Unification Algorithm

- 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 φ_1 and φ_2 be two atomic expression sentences and σ be a unifier such that, $\varphi_1\sigma = \varphi_2\sigma$, then it can be expressed as UNIFY (φ_1, φ_2).

② Resolution as Proof Procedure

Resolution is a theorem proving technique that proceeds by building refutation proofs, i.e. proofs by contradictions. It was invented by mathematician John Alan Robinson in the year 1965.

Resolution is used, if there are various statements given and we need to prove conclusion of those statements.

Unification is a key concept in proofs by resolutions.

Resolution is a single inference rule which can efficient operate on the conjunctive normal form or clausal fo

Scanned with CamScanner

Clause : Disjunction of literals is called a clause.

Conjunctive Normal Form : A sentence represented as a conjunction of classes is said to be conjunctive normal form or CNF.

Input : Two Literals L1 and L2.
Output : A set of substitutions
Algorithm : Unification

```
Step 1: If $\Psi_1$ or $\Psi_2$ is a variable or constant then:
    a) If $\Psi_1$ or $\Psi_2$ are identical then return NIL
    b) Else if $\Psi_1$ is a variable
        a. then if $\Psi_1$ occurs in $\Psi_2$, then return FAILURE.
        b. Else return $\{(\Psi_2/\Psi_1)\}$
    c) Else if $\Psi_2$ is a variable
        a. If $\Psi_2$ occurs in $\Psi_1$, then return FAILURE.
        b. Else return $\{(\Psi_1|\Psi_2)\}$
    d) Else return FAILURE.
Step 2: If the initial Predicate symbol in $\Psi_1$ and $\Psi_2$ are
        not same, then return FAILURE.
Step 3: If $\Psi_1$ and $\Psi_2$ have a different number of
        arguments, then return FAILURE.
Step 4: Set submission set (SUBST) to NIL.
Step 5: For i=1 to the number of elements in $\Psi_1$.
    a) call unify function with the ith element of $\Psi_1$
       and the ith element of $\Psi_2$ and put the result into S.
    b) If S= failure then returns failure.
    c) If S ≠ NIL then do,
```

Scanned with CamScanner

a. Apply S to the remainder of both L1 and L2.
b. SUBST = APPEND (S, SUBST).

Step6. Return SUBST.

FAQs

1) Why resolution is required?

→ Resolution is used if there are various statements given and we need to prove a conclusion of those statements. Unification is a key ~~po~~ concept in proofs by resolutions. Resolution is a single inference rule which can efficienty operate on the conjunctive normal form or clausal form.

2) What are the pre-requisites for applying unification algorithm?

→ · Predicate symbol must be same, atoms or expression with different predicate symbol can never be unified.

· Number of arguments in both expressions must be identical.

· unification will fail if there are two similar variables present in the same expression.

3) What are the applications of unification algorithm?

→ Syntactical first-order unification is used in logic programming and programming language type system implementation, especially in Hindley-Milner based type inference algorithms. Semantic unification is used in SMT solvers, term rewriting algorithms and cryptographic protocol analysis.

Scanned with CamScanner

AI LAB ASSIGNMENT NO.4
UNIFICATION ALGORITHM

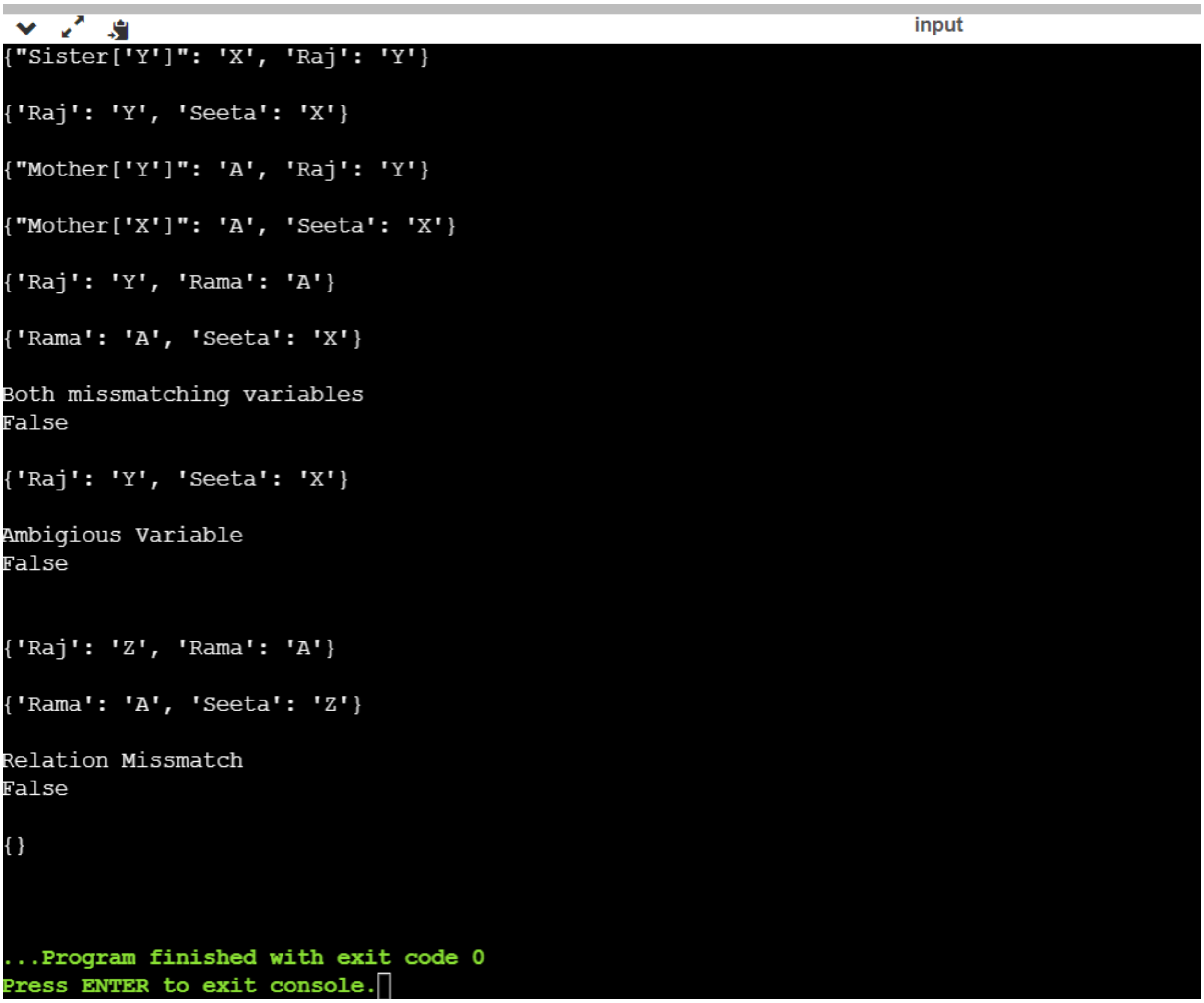
NAME : KETAKI PATIL
ROLL NO : PA-17
BATCH : A1

CODE :

```
/*
Unification Algorithm
AI LAB Assignment 4
Name: Ketaki Patil
Roll No. PA 17
Batch 1
*/

import random
class Variable:
    def __init__(self,value):
        self.value = value
    def __eq__(self, other):
        return self.value == other.value
class Constant:
    def __init__(self,value):
        self.value = value
    def __eq__(self, other):
        return self.value == other.value
class Rel:
    def __init__(self,name,args):
        #This is a list
        self.name = name
        self.value = str(self.name)+str([i.value for i in args])
        self.args = args



def Unify(L1,L2,testset):
    '''
    L1 and L2 are Rel types, variables or constants
    '''
    #If both are variable or constants
    if(isinstance(L1,Variable) or isinstance(L2,Variable) or
isinstance(L1,Constant) or isinstance(L2,Constant)):
    if L1 == L2:
        return None
    elif isinstance(L1,Variable):
        if isinstance(L2,Variable):
            print("Both missmatching variables")
            return False
        else:
            if L1.value not in testset.values():
                return [L2,L1]
            else:
                print("Ambigious Variable")
                return False
    elif isinstance(L2,Variable):
        if isinstance(L1,Variable):
            print("Both missmatching variables")
            return False
        else:
            if L2.value not in testset.values():
                return [L1,L2]
            else:
                print("Ambigious Variable")
                return False
    else:
        print("Missmatch")
        return False

#Ensuring the functions are the same
elif L1.name != L2.name:
    print("Relation Missmatch")
    return False
#Ensuring the functions have the same number of arguments
elif len(L1.args) != len(L2.args):
    print("length does not match")
    return False

SUBSET = {}

for i in range(len(L1.args)):
    S = Unify(L1.args[i],L2.args[i],SUBSET)
    if S==False:
        return False
    if S != None:
        SUBSET[S[0].value] = S[1].value

return SUBSET
```

```
if __name__ == "__main__":


print(Unify(Rel("Knows",[Constant("Raj"),Variable("X")]),Rel("Knows",[
Variable("Y"),Rel("Sister",[Variable("Y")])]),{}))
    print()

print(Unify(Rel("Knows",[Constant("Raj"),Variable("X")]),Rel("Knows",[
Variable("Y"),Constant("Seeta")]),{}))
    print()

print(Unify(Rel("Knows",[Constant("Raj"),Variable("A")]),Rel("Knows",[
Variable("Y"),Rel("Mother",[Variable("Y")])]),{}))
    print()

print(Unify(Rel("Knows",[Constant("Seeta"),Variable("A")]),Rel("Knows"
,[Variable("X"),Rel("Mother",[Variable("X")])]),{}))
    print()

print(Unify(Rel("Knows",[Constant("Raj"),Variable("A")]),Rel("Knows",[
Variable("Y"),Constant("Rama")]),{}))
    print()

print(Unify(Rel("Knows",[Constant("Seeta"),Variable("A")]),Rel("Knows"
,[Variable("X"),Constant("Rama")]),{}))
    print()

print(Unify(Rel("Mother",[Variable("Y"),Variable("A")]),Rel("Mother",[
Variable("X"),Variable("A")]),{}))
    print()

print(Unify(Rel("Knows",[Constant("Raj"),Variable("X")]),Rel("Knows",[
Variable("Y"),Constant("Seeta")]),{}))
    print()

print(Unify(Rel("Knows",[Constant("Raj"),Variable("Y")]),Rel("Knows",[
Variable("Y"),Constant("Seeta")]),{}))
    print()
    print()

print(Unify(Rel("Knows",[Constant("Raj"),Variable("A")]),Rel("Knows",[
Variable("Z"),Constant("Rama")]),{}))
    print()

print(Unify(Rel("Knows",[Constant("Seeta"),Variable("A")]),Rel("Knows"
,[Variable("Z"),Constant("Rama")]),{}))
    print()
print(Unify(Rel("Mother",[Variable("Y"),Variable("A")]),Rel("Daughter"
,[Variable("X"),Variable("A")]),{}))
    print()

print(Unify(Rel("Mother",[Variable("Z"),Variable("A")]),Rel("Mother",[
Variable("Z"),Variable("A")]),{}))
    print()
```

OUTPUT :



```
{"Sister['Y']": 'X', 'Raj': 'Y'}

{'Raj': 'Y', 'Seeta': 'X'}

{"Mother['Y']": 'A', 'Raj': 'Y'}

{"Mother['X']": 'A', 'Seeta': 'X'}

{'Raj': 'Y', 'Rama': 'A'}

{'Rama': 'A', 'Seeta': 'X'}

Both missmatching variables
False

{'Raj': 'Y', 'Seeta': 'X'}

Ambigious Variable
False


{'Raj': 'Z', 'Rama': 'A'}

{'Rama': 'A', 'Seeta': 'Z'}

Relation Missmatch
False

{}



...Program finished with exit code 0
Press ENTER to exit console.
```