ARTIFICIAL INTELLIGENCE (18CSC305J) LAB

EXPERIMENT 7

Implementation of unification and resolution for real world problems using C language

Harsh Goel
RA1811003010185
CSE-C1

Aim:

To implement unification and resolution for real world problems.

Problem Description:

Unification can be seen as a generalization of pattern matching, so let's start with that first.

We're given a constant term and a pattern term. The pattern term has variables. Pattern matching is the problem of finding a variable assignment that will make the two terms match. For example:

- Constant term: f(a, b, bar(t))
- Pattern term: f(a, V, X)

Trivially, the assignment V=b and X=bar(t) works here. Another name to call such an assignment is a substitution, which maps variables to their assigned values. In a less trivial case, variables can appear multiple times in a pattern:

- Constant term: f(top(a), a, g(top(a)), t)
- Pattern term: f(V, a, g(V), t)

Here the right substitution is V=top(a).

Sometimes, no valid substitutions exist. If we change the constant term in the latest example to f(top(b), a, g(top(a)), t), then there is no valid substitution becase V would have to match top(b) and top(a) simultaneously, which is not possible.

Problem Formulation:

Unification is just like pattern matching, except that both terms can contain variables. So we can no longer say one is the pattern term and the other the constant term. For example:

- First term: f(a, V, bar(D))
- Second term f(D, k, bar(a))

Given two such terms, finding a variable substitution that will make them equivalent is called unification. In this case the substitution is {D=a, V=k}.

At first we will be comparing the values given for substitution, for comparison we will be having 3 cases

- When arguments are same
- When arguments are not same
- And , when arguments are identical

Other than the first case we do not need to make substitution for the other two cases.

ALGORITHM:

Step.1: Initialize the substitution set to be empty.

Step.2: Recursively unify atomic sentences:

a) Check for Identical expression match.

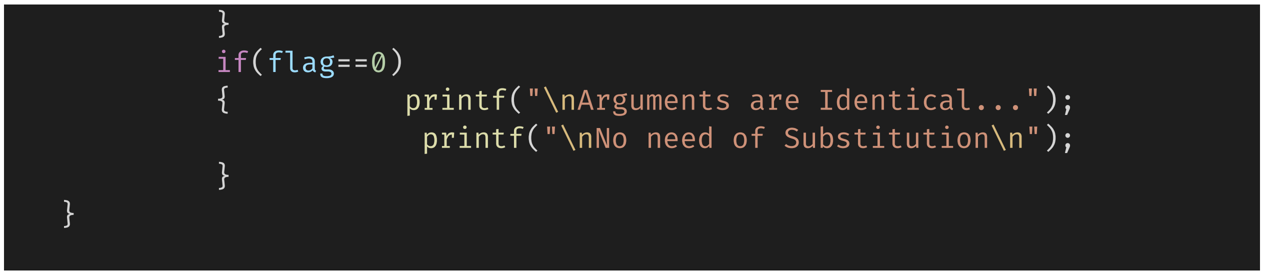
b) If one expression is a variable vi, and the other is a term ti which does not contain variable vi, then:

1. Substitute ti / vi in the existing substitutions
2. Add ti /vi to the substitution setlist.
3. If both the expressions are functions, then function name must be similar, and the number of arguments must be the same in both the expression.

Source Code:

Language- **C**

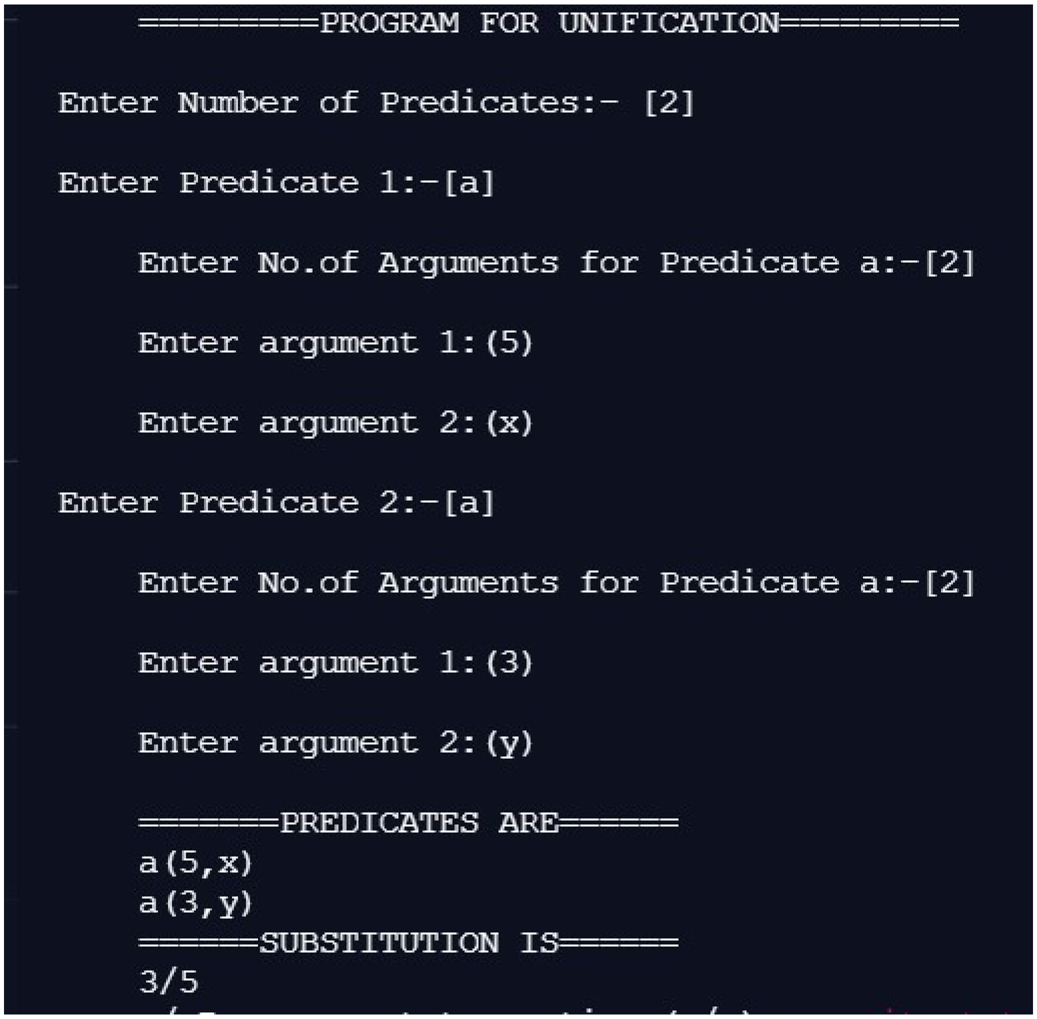
```c
#include<stdio.h>
int no_of_pred;
int no_of_arg[10];
int i,j;
char nouse;
char predicate[10];
char argument[10][10];
void unify();
void display();
void chk_arg_pred();
   void main()
   {
   char ch;
   do{
            printf("\t=========PROGRAM FOR UNIFICATION=========\n");
            printf("\nEnter Number of Predicates:- [ ]\b\b");
            scanf("%d",&no_of_pred);

            for(i=0;i<no_of_pred;i++)
            {
            scanf("%c",&nouse);    //to accept "enter" as a character
            printf("\nEnter Predicate %d:-[ ]\b\b",i+1);
            scanf("%c",&predicate[i]);
            printf("\n\tEnter No.of Arguments for Predicate %c:-
[ ]\b\b",predicate[i]);
            scanf("%d",&no_of_arg[i]);
                        for(j=0;j<no_of_arg[i];j++)
                        {
                         scanf("%c",&nouse);
                         printf("\n\tEnter argument %d:( )\b\b",j+1);
                         scanf("%c",&argument[i][j]);
                        }
            }
            display();
            chk_arg_pred();
            printf("Do you want to continue(y/n):   ");
            scanf("%c",&ch);
       }while(ch == 'y');
   }
void display()
   {
       printf("\n\t======PREDICATES ARE======");
            for(i=0;i<no_of_pred;i++)
            {
             printf("\n\t%c(",predicate[i]);
                        for(j=0;j<no_of_arg[i];j++)
                        {
                        printf("%c",argument[i][j]);
                        if(j!=no_of_arg[i]-1)
                                    printf(",");
                        }
             printf(")");
            }
   }
   void chk_arg_pred()
   {
   int pred_flag=0;
   int arg_flag=0;

            for(i=0;i<no_of_pred-1;i++)
            {
                        if(predicate[i]!=predicate[i+1])
                        {
                        printf("\nPredicates not same..");
                        printf("\nUnification cannot progress!");
                        pred_flag=1;
                        break;
                        }
            }
     if(pred_flag!=1)
     {
            for(i=0;i<no_of_arg[i]-1;i++)
            {
                        if(no_of_arg[i]!=no_of_arg[i+1])
                        {
                        printf("\nArguments Not Same..!");
                        arg_flag=1;
                        break;
                        }
            }
     }
            if(arg_flag==0&&pred_flag!=1)
                        unify();

   }
   void unify()
   {
            int flag=0;
            for(i=0;i<no_of_pred-1;i++)
            {
                 for(j=0;j<no_of_arg[i];j++)
                 {
                        if(argument[i][j]!=argument[i+1][j])
                        {
                          if(flag==0)
                          printf("\n\t======SUBSTITUTION IS======");
                        printf("\n\t%c/%c",argument[i+1][j],argument[i][
j]);

                         flag++;
                        }
                 }
```



```
            }
            if(flag==0)
            {          printf("\nArguments are Identical...");
                        printf("\nNo need of Substitution\n");
            }
   }
```

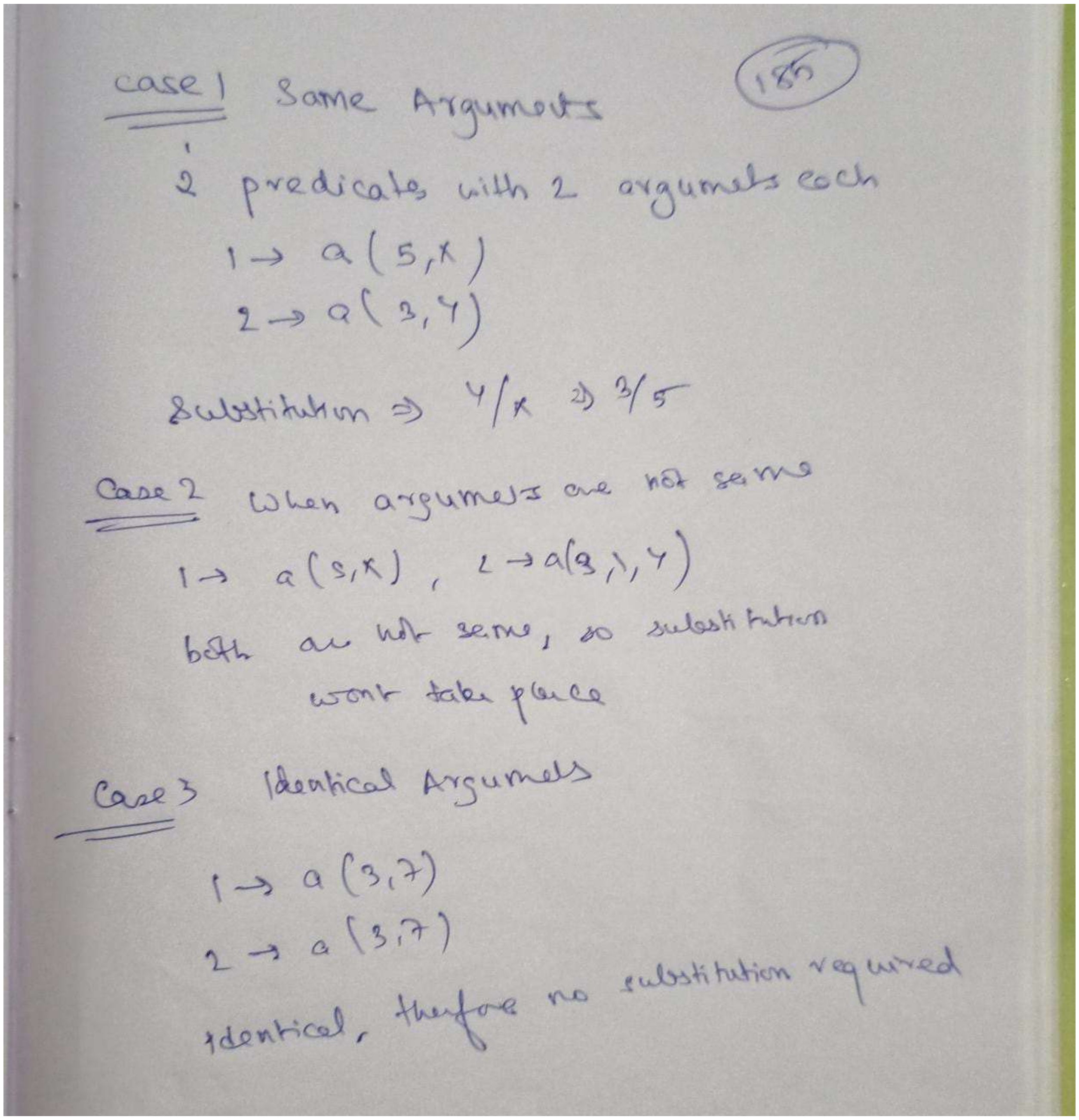
TEST CASE:

Input from User and Output from User



```
     ==========PROGRAM FOR UNIFICATION==========

  Enter Number of Predicates:- [2]

  Enter Predicate 1:-[a]

      Enter No.of Arguments for Predicate a:-[2]

      Enter argument 1:(5)

      Enter argument 2:(x)

  Enter Predicate 2:-[a]

      Enter No.of Arguments for Predicate a:-[2]

      Enter argument 1:(3)

      Enter argument 2:(y)

      =======PREDICATES ARE======
      a(5,x)
      a(3,y)
      ======SUBSTITUTION IS======
      3/5
```

Verification:



case 1 Same Arguments

185

2 predicate with 2 arguments each

1 → a(5,x)

2 → a(3,y)

Substitution ⇒ y/x 2) 3/5

Case 2 When arguments are not same

1 → a(5,x) , 2 → a(3,1,y)

both are not same, so substitution won't take place

Case 3 Identical Arguments

1 → a(3,7)

2 → a(3,7)

identical, therefore no substitution required

Result: Hence, successfully implemented implement unification and resolution for real world problems and verified test cases.