ASSIGNMENT NO.

AIM : Implementation of Unification algorithm.

OBJECTIVE :

- To understand Unification Algorithm.
- To implement Unification Algorithm.

SOFTWARE REQUIREMENTS :

- Linux Operating System
- C Compiler

MATHEMATICAL MODEL :

Consider a set S consisting of all elements related to a program. The mathematical model is given as below,

S={s,e,X,Y,fme,DD,NDD,MemShared}

Where,

s= Initial State
e= End State
X=Input Given fX1,X2,X3,X4g where, X1=No.of Predicates,
X2= Predicates, X3=No.of Arguments, X4= Arguments
Y=Output Obtained fY1,Y2g
where, Y1=Predicates, Y2=Substitution
fme= Function used fvoid unify(); void display(); void chkargpred();g
DD=Deterministic Data
NDD=Non-Deterministic Data
MemShared=Processor core used by program.

THEORY :

1. Unification Algorithm:

Unification, in computer science and logic, is an algorithmic process of solving equations between symbolic expressions. Depending on which expressions (also called terms) are allowed to occur in an equation set (also called unification problem), and which expressions are considered equal, several frameworks of unification are distinguished. If higher- order variables, that is, variables representing functions, are allowed in an expression, the process is called higher order unification, otherwise first-order unification.

If a solution is required to make both sides of each equation literally equal, the process is called syntactical unification, otherwise semantical, or equational unification, or E-unification, or unification modulo theory. A solution of a unification problem is denoted as a substitution, that is, a mapping as signing a symbolic value to each variable of the problem's expressions. A unification algorithm should compute for a given problem a complete and minimal substitution set, that is, a set covering all its solutions, and containing no redundant members. Depending on the framework, a complete and minimal substitution set may have at most one, at most finitely many, or
Possibly infinitely many members, or may not exist at all.

Unification:

Unfication is a pattern-matching procedure Takes two atomic sentences, called literals, as input and Returns Failure if they do not match and a substitution list,$, if they do That is, unify(p,q) =$ means subst($, p) = subst($, q) for two atomic sentences, p and q,$ is called the most general unifier.

- All variables in the given two literals are implicitly universally quanti_ed
- To make literals match, replace (universally quanti_ed) variables by terms

Lifted inference rules require _nding substitutions that make di_erent logical expressions look identical. This process is called uni_cation and is a key component of all _rst-order inference algorithms. The UNIFY algorithm takes tcvo sentences and returns a uni_er for them if one exists: UNIFY(p,q).

Example: Suppose we have a query Knows (John, x) : whom does John know? Some answers to this query can be found by _nding all sentences in the knowledge base that unify with Knows (John, x) . Here are the results of uni_cation with four di_erent sentences that might be in the lcnowledge base.

UNIFY(Knows(John,x), Knows(John, Jane))= (X, Jane)
UNIFY(Knows(John,x), Knows(y, Bill))= (x,Bill) (y, John)
UNIFY(Knows(John,x), Knows(y, other)) = (y, John) (x, Mother(John))
UNIFY (Knows(John, x) , Knows (x , Elizabeth)) = Jail.

The last uni_cation fails because x cannot take on the values John and Elizabeth at the same time. Now, remember that Knows(x, Elizabeth) means "Everyone knows Elizabeth," so we should be able to infer that John knows Elizabeth. The problem arises only because the two sentences happen to use the same variable name x. The problem can be avoided by standardizing

apart one of the two sentences being unified, which means renaming its APART variables to avoid name clashes. For example, we can re-
name x in Knows(x, Elizabeth) to z(a new variable name) without changing its meaning. Now the unification will work: UNIFY(Knows(John,x) ,Knows(z,Elizabeth)) = (z|Elizabeth, z|John).

ALGORITHM:

```
procedure unify(p, q, )

Scan p and q left-to-right and _nd the _rst corresponding

terms where p and q disagree (i.e., p and q not equal)

If there is no disagreement, return @(success!)

Let r and s be the terms in p and q; respectively;

where disagreement first occurs

If variable(r) then
{
        Let@ = union(@; fr=sg)
        Return(unify(subst(@, p), subst(@; q); @))
}
else if variable(s) then
{
        Let@ = union(@; fs=rg)
        Return unify(subst(@, p), subst(@; q);@)
}
else return Failure
end
```

CONCLUSION:

Thus we have studied and Implemented Uni_cation Algorithm.