UNIFICATION

Unification is the process of substituting suitable values to the predicates in the predicate logic. It is used in Resolution., that is resolving the proof.

In propositional logic it is easy to determine that two literals cannot both be true at the same time. L and ¬L in predicate logic, this matching process is complicated since the arguments of the predicates must be considered.

Example: Man (John) and ¬ man (John) is a contradiction, while man(John) and ٦man(spot) is not. For contradictions, need a matching procedure that compares two literals and discovers whether there exists a set of substitutions that makes them identical. The recursive procedure called the unification algorithm.

1. Unification It is very simple. To unify two literals, first check if their first elements are same. If so proceed. Otherwise, no way they can be unified, regardless if their arguments.

Example The two literals try assassinate (Marcus, Caesar) hate (Marcus ,Caesar) cannot be unified.

- If predicate symbols match, check the arguments, one pair at a time.
- If the first matches, we can continue with the second and so on.
- To test each argument pair, simply call the unification procedure recursively.
- Matching rules are simple identical constants or predicates can match.

A variable can match another variable, any constant or a predicate expression, with the restriction that the predicate expression must not contain any instances of the variable being matched.

2. Complication : We must find a single, consistent substitution for the entire literal, not separate ones for each piece of it. Each substitution is applied to the remainder of the literals before we unify them.

Example: Unify the expressions P(x,x), P(y,z)

The two instances of P match fine. Next we compare x and y and decide that if we substitute y for x, they could match.

We will write that substitution as y/x We could, of course, have decided instead to substitute x for y, since they are both just dummy variable names.

The algorithm will simply pick one of these two substitutions. But now, if we simply continue and match x and z. We produce the substitution z/x. But we cannot substitute both y and z for x, so we have not produced a consistent substitution. What we need to do after finding the first substitution y/x is to make that substitution z/y.

The entire unification process has now succeeded with a substitution that is the composition of the two substitutions we found. we write the composition as

(z/y) (y/x) following standard notation for function composition.

In general, the substitution

(a1/a2, a3/a4,..) (b1/b2, b3/b4, ..) .. means to apply all the substitutions of the right most list, then take the result and apply all the ones of the next list, and so forth, until all substitutions have been applied.

The object of the unification procedure is to discover at least one substitution that cause two literals to match. Usually, if there is one such substitution there are many.

For example, the literals **hate (x,y) hate (Marcus, z)** Could be unified with any of the following substitutions:

(Marcus/x, z/y)

(Marcus/x, y/z)

(Marcus/x, Caesar/y, Caesar/z)

(Marcus/x, Polonius/y, Polonius/z)

The first two from above substitutions are equivalent except for lexical variation. The second two, they produce a match, produce a substitution that is restrictive than absolutely necessary for the match. Final substitution used by the resolution procedure, it is useful to generate the most general unifier possible.

Procedure Unify (L1, L2) –

Description:

● It returns as its value a list representing the composition of the substitutions that were performed during the match.

● The empty list, NIL, indicates that a match was found without any substitutions

● The single value FAIL indicates that the unification procedure failed. 3. Algorithm: Unify(L1, L2)

Step 1 : If L1 or L2 are both variables or constants, then:

i. If L1 and L2 are identical, then return NIL.
ii. Else if L1 is a variable, then if L1 occurs in L2 then return {FAIL}, else return (L2/L1).
iii. Else if L2 is a variable, then if L2 occurs in L1 then return {FAIL} , else return (L1/L2)
iv. Else return {FAIL}.

Step 2 : If the initial predicate symbols in L1 and L2 are not identical, then return {FAIL}.

Step 3 : If LI and L2 have a different number of arguments, then return {FAIL}.

Step 4 : Set SUBST to NIL. (At the end of this procedure, SUBST will contain all the substitutions used to unify L1 and L2.)

Step 5 : For i ← 1 to number of arguments in L1 :

i. Call Unify with the ith argument of L1 and the ith argument of L2, putting result in S.
ii. If S contains FAIL then return {FAIL}.
iii. If S is not equal to NIL then: (a) Apply S to the remainder of both L1 and L2. (b) SUBST: = APPEND(S, SUBST).
iv. Return SUBST. The only part of this algorithm that we have not yet discussed is the check in steps 1(b) and 1(c) to make sure that an expression involving a given variable is not unified with that variable.

Suppose we were attempting to unify the expressions f(x,x) f(g(x),g(x)) If we accepted g(x) as a substitution for x, then we would have to substitute it for x in the remainder of the expressions. But this leads to infinite recursion since it will never be possible to eliminate x