Substitutions and Unifiers in First-Order Predicate Logic

A motivating example:

Axioms: (∀x)(Bird(x) →Flies(x))
Bird(Tweety)
Goal: Flies(Tweety)

Convert to a clausal form database:

Axioms: ¬Bird(x) ∨ Flies(x)
Bird(Tweety)
Negated goal: ¬Flies(Tweety)

Without further operations, no resolution is possible.

The needed operation is *substitution.*

$$\neg\text{Bird}(x) \vee \text{Flies}(x) \xrightarrow{\{\text{Tweety}/x\}} \neg\text{Bird}(\text{Tweety}) \vee \text{Flies}(\text{Tweety})$$

The notation Tweety/x means
"Substitute Tweety for x."

To employ resolution for first-order logic, it is necessary to develop substitution in a systematic manner.

A more complex example:

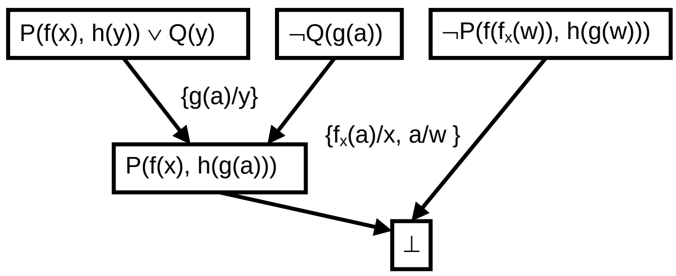
Axioms: (∀x)(∀y)(P(f(x), h(y)) ∨ Q(y))
(∀x)(¬Q(g(a)))
Goal: (∃y)(∀x)P(f(x),h(g(y)))

Note: a, b, c denote constants
x, y , z, w denote variables.

Negate the goal and normalize all:

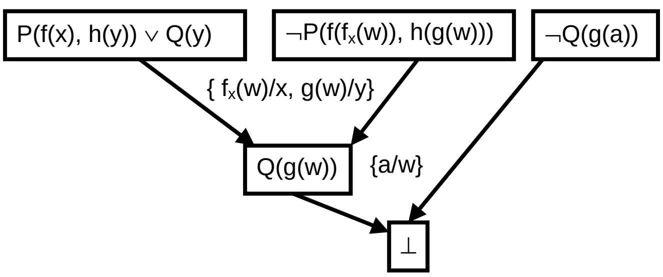
P(f(x), h(y)) ∨ Q(y)
(¬Q(g(a)))
¬P(f(f_x(w)),h(g(w)))

Now perform resolution:

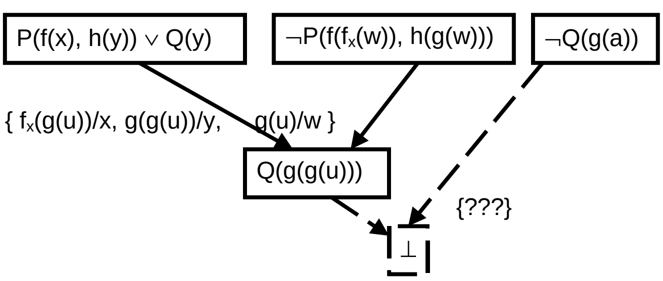


Notice that some fairly complex decisions regarding which substitutions to make are necessary.

The technique should also work if resolution is performed in another order.



Note that a substitution which is too specific can cause a problem.



Thus, it is necessary to investigate this substitution issue thoroughly.

Substitution and Unification:

- Unification is the operation which is applied to terms in order to make them "match" so that resolution can be performed.
- It is accomplished by applying *substitutions* to the clauses containing the atoms to be matched.

We now investigate these ideas in more detail.

Notational convention: Throughout this discussion, it is assumed that there is an extant first-order logic $\mathcal{L} = (\mathcal{R}, \mathcal{C}, \mathcal{A}, \mathcal{T})$, with $\mathcal{T} = (\mathcal{V}, \mathcal{K}, \mathcal{F})$.

In general: a, b, c denote constants;
x, y, z, w denote variables;
P, Q, R, S denote predicate letters;
f, g, h denote function symbols;
unless stipulated to the contrary.

Substitution:

Definition: A *substitution* is a finite set of specifications of the form

$$t/v$$

in which t is a term and v is a variable. Substitutions are usually written in set notation:

$$\{t_1/v_1, t_2/v_2, .., t_n/v_n\}$$

Substitutions are applied to terms, or to sets of terms.

Important: The semantics of a substitution is that all of its elements are applied simultaneously.

Example: The application of the substitution

$$\{g(y)/x, h(z)/y, x/z\}$$

to

$$f(x, y, g(z), w)$$

is

$$f(g(y), h(z), g(x), w),$$

and not

$$f(g(h(x)), h(x), g(x), w).$$

The order of the elements in a substitution list is irrelevant.

Note also that the substitution need not specify a replacement for each variable in the formula. Variables not listed in the substitution are left unchanged.

Notation: The symbol σ (and subscripted versions threreof) are typically used to represent substitutions. The application of a substitution σ to a term t is denoted

$$t\sigma.$$

Substitutions may also be applied to atoms. In that case, the substitution is applied to each term in the atom.

Example: Let $\varphi = P(f(x,y), g(h(y)), z, w)$
$\sigma = \{h(y)/x, a/y, w/z\}$

Then $\varphi\sigma = P(f(h(y),a), g(h(a)), w, w)$

Substitutions may furthermore be applied to entire clauses. In this case, the substitution is applied to each atom of the clause.

Example: Let $\varphi = P(f(x,y), g(h(y)), z, w) \vee Q(y,z)$
$\sigma = \{h(y)/x, a/y, w/z\}$

Then $\varphi\sigma = P(f(h(y),a), g(h(a)), w, w) \vee Q(a,w)$.

Note particularly the "right" notation, which differs from the more traditional mathematical $\sigma(\varphi)$.

Composition of substitutions:

Substitutions may be composed.

Example: Let
$\sigma_1 = \{f(a)/x, g(b,z)/y, x/z\}$
$\sigma_2 = \{w/x, h(z)/y, a/z\}$

Then $\sigma_1\sigma_2 = \{f(a)/x, g(b,a)/y, w/z\}$

Note that

- Substitution composition occurs from left to right. Thus, $\sigma_1\sigma_2$ means that first σ_1 should be applied, and then σ_2.

- Application of substitution respects composition. That is:

$$\varphi(\sigma_1\sigma_2) = (\varphi\sigma_1)\sigma_2$$

Example: Let $\varphi = P(x,y,z)$, and let σ_1 and σ_2 be as above.

Then $\varphi\sigma_1 = P(f(a), g(b,z), x)$
$(\varphi\sigma_1)\sigma_2 = P(f(a), g(b,a), w) = \varphi(\sigma_1\sigma_2)$

Note, however, that composition is not commutative:

$$\sigma_2\sigma_1 = \{w/x, h(x)/y, a/z\} \neq \sigma_1\sigma_2.$$

Also note that a substitution is not necessarily the composition of its components.

Example: Let $\sigma_1 = \{f(a)/x, g(b,z)/y, x/z\}$ as above.

Let $\sigma_{11} = \{ f(a)/x \}$; $\sigma_{12} = \{g(b,z)/y\}$; $\sigma_{13} = \{x/z\}$.

Then $\sigma_{11}\sigma_{12}\sigma_{13} = \{f(a)/x, g(b,x)/y, x/z\} \neq \sigma_1$.

The result even depends upon the ordering:

$$\sigma_{13}\sigma_{12}\sigma_{11} = \{f(a)/z, g(b,z)/y\} \neq \sigma_{11}\sigma_{12}\sigma_{13}.$$

Ordering of substitutions:

Definition: Let σ_1 and σ_2 be substitutions. Write

$$\sigma_1 \preccurlyeq \sigma_2$$

just in case there is a substitution σ such that

$$\sigma_1 = \sigma_2\sigma.$$

In this case, it is said that σ_2 is *more general* than σ_1.

Example: Let $\sigma_1 = \{f(a)/x, a/y\}$.
$\sigma_2 = \{f(a)/x\}$

Then $\sigma_1 \preccurlyeq \sigma_2$ since $\sigma_1 = \sigma_2\sigma$ for
$$\sigma = \{a/y\}$$
has the property that
$$\sigma_1 = \sigma_2\sigma.$$

Caution: This definition can be misleading.
Example: Let $\sigma_1 = \{f(a)/x\}$
$\sigma_2 = \{f(y)/x\}$.
It might appear at first that
$$\sigma_1 \preccurlyeq \sigma_2$$
with $\sigma = \{a/y\}$ yielding $\sigma_1 = \sigma_2\sigma$.
This is not the case! Try it on the formula
$$P(x,y)$$
and see what happens.

Some further useful ideas, without proof:

Definition: A substitution σ is a *renaming* if it defines a permutation of the some set of variables. For example, {x/y, z/x, y/z} is a renaming.

Definition: Two substitutions σ_1 and σ_2 are *equivalent* if there is a renaming σ such that
$$\sigma_1 = \sigma_2\sigma.$$

In this case, there must also be a renaming σ' such that $\sigma_2 = \sigma_1\sigma'$.

Fact: If
$$\sigma_1 \preccurlyeq \sigma_2$$
and
$$\sigma_2 \preccurlyeq \sigma_1$$
both hold, then there are renamings σ and σ' such that
$$\sigma_1 = \sigma_2\sigma$$
and
$$\sigma_2 = \sigma_1\sigma'. \square$$

Unification:

Definition: Let ψ_1 and ψ_2 be atoms. A *unifier for* ψ_1 and ψ_2 is a substitution σ such that
$$\psi_1\sigma = \psi_2\sigma$$

Example: Let $\psi_1 = Bird(x)$
$\psi_2 = Bird(Tweety)$.

Then $\sigma = \{Tweety/x\}$ is a unifier for these atoms.

Example: Let $\psi_1 = P(a, x, f(g(y))$
$\psi_2 = P(z, f(z), f(w))$

Then $\sigma = \{a/z, f(a)/x, g(y)/w\}$ is a unifier for ψ_1 and ψ_2.

Definition: A unifier for atoms ψ_1 and ψ_2 is a *most general unifier (mgu)* if it is a most general substitution which unifies ψ_1 and ψ_2.

Example: Both examples above are mgu's.

Theorem: If two atoms ψ_1 and ψ_2 have a unifier, then they have a most general unifier. Furthermore, there is an algorithm which can determine whether or not two atoms are unifiable, and, if so, deliver an mgu for them. $\square$

The mgu algorithm:

Before presenting the algorithm formally, it will be illustrated on some examples.

Example: Let $\psi_1 = P(a, x, f(g(y)))$
$\psi_2 = P(z, f(z), f(w))$

Step 1: Make sure that the predicate symbols match. Atoms with different predicate symbols can never be unified.

Step 2: Attempt to unify each pair of terms.

- The first pair is (a,z). Since one of the elements is a variable, they can be unified by substituting the other term for this variable. The appropriate substitution is a/z. So, set
$$mgu \leftarrow \{a/z\}$$
This substitution must also be applied to both clauses yielding
$$P(a, x, f(g(y)))$$
$$P(a, f(a), f(w))$$

- The second pair is (x,f(a)). Again, since one term is a variable, substitute the other for it: f(a)/x. The new value of mgu is the old value, composed with this new substitution.
$$mgu \leftarrow mgu \circ \{f(a)/x\} = \{a/z, f(a)/x\}$$
This substitution must also be applied to both clauses yielding
$$P(a, f(a), f(g(y)))$$
$$P(a, f(a), f(w))$$

- The third and final pair is (f(g(y)),f(w)). Neither is an atom, so we check to see whether the function symbols are the same. They are, so we strip them and unify each pair of sub-terms. (In this case, there is just one such pair.) The new pair is (g(y),w). This pair may be unified with the substitution g(y)/w. Thus,
  mgu ← mgu∘{g(y)/w} = {a/z, f(a)/x, g(y)/w}
  This substitution must also be applied to both clauses yielding
  P(a, f(a), f(g(y)))
  P(a, f(a), f(g(y)))
  The clauses match, and an mgu has been found.

Note: The order in which the terms are unified does not matter.

Starting with ψ_1 = P(a, x, f(g(y)))
ψ_2 = P(z, f(z), f(w))
again, let us unify the second pair of terms first.

This yields {f(z)/x}, with resulting atoms
P(a, f(z), f(g(y)))
P(z, f(z), f(w))

Now unify the third pair of terms. The mgu is {g(y)/w}, so after this step the unifier is {f(z)/x, g(y)/w}, and the atoms are
P(a, f(z), f(g(y)))
P(z, f(z), f(g(y)))

Finally, we unify the first pair of terms, using a/z. The final mgu is {f(z)/x, g(y)/w, a/z}, and the final terms match, as before:
P(a, f(a), f(g(y)))
P(a, f(a), f(g(y)))

Not all pairs of atoms unify, of course. Here are some examples of failure.

Example: ψ_1 = Q(f(a), g(x))
ψ_2 = Q(y, y)

To unify the first pair, (f(a), y), the substitution f(a)/y is used. The atoms become
Q(f(a), g(x))
Q(f(a), f(a))

Now, the second pair is (g(x), f(a)). Since the function symbols are different, unification fails.

Suppose that we try to unify the second pair first. In that case, the pair is (g(x), y), which is unifiable with g(x)/y. The atoms become
Q(f(a), g(x))
Q(g(x), g(x))

Fact: If unification fails for one order of the pairs, then it will fail for all orders. The order of attempt will not affect the result.

The occurs check:

There is a rather subtle but nonetheless important point which must be observed.

Example: ψ_1 = Q(x, x)
ψ_2 = Q(y, f(y))

Unifying the first pair using y/x, we get
Q(y, y)
Q(y, f(y))

The second pair, (y, f(y)), is strange in that both components involve y. Unless they are identical, it is impossible to unify them.

To detect this situation requires a special test called the *occurs check*, which tests whether or not a given variable occurs in a given term.

The formal algorithm:

Basic data types:
Term:
Substitution:
List_of_terms: $\langle t_1, t_2, .., t_n\rangle$
Logical_atom: Something like P(x,y,g(a,x))

Basic functions:
Is_variable(x: term): Returns Boolean.
True if the term is a variable.

Is_constant(x: term): Returns Boolean.
True if the term is a constant symbol

Is_functional_term(x: term): Returns Boolean.
True if the term is of the form $f(t_1, t_2, .., t_n)$.

Function_symbol(x: term): Returns the function symbol of a functional term: $f(t_1, t_2, .., t_n) \mapsto f$.

Term_list(x: term): Returns list_of_terms.
$f(t_1, t_2, .., t_n) \mapsto \langle t_1, t_2, .., t_n\rangle$

Compose_substitutions(σ_1, σ_2):
Returns substitution.

First(x:list_of_terms): $\langle t_1, t_2, .., t_n\rangle \mapsto t_1$

Rest(x): Returns list_of_term.
$\langle t_1, t_2, .., t_n\rangle \mapsto \langle t_2, .., t_n\rangle$
Arglist(x:Logical_atom): Returns: list_of_term
$P(t_1, t_2, .., t_n) \mapsto \langle t_1, t_2, .., t_n\rangle$


```
Procedure Term_mgu (s, t: term; σ: substitution);
                              Returns: substitution;
--- If s and t are unifiable,
--- returns the composition of σ with their unifier.
--- If s and t are not unifiable, returns FAIL.
 Begin
    Do_first_true_conditional:
       Is_variable(s):
          Return variable_mgu(s, t, σ);
       Is_variable(t):
          Return variable_mgu(t, s, σ);
       Is_constant(s):
           If (Is_constant(t) ∧ s=t)
               then return σ else return FAIL;
       Is_functional_term(s):
           If  (Is_functional_term(t) ∧
                function_symbol(s) =
                              function_symbol(t))
               then return
                  Term_list_mgu (Term_list(s),
                                      Term_list(t), σ);
               else return FAIL;
    End Do_first_true_conditional;
End Procedure; {Term_mgu}
```



```
Procedure Variable_mgu
                  (s: variable; t: term: σ: substitution);
                   Returns: substitution;
--- If s does not occur in t, returns σ∘{t/s}.
--- If s occurs in t, returns FAIL.
Begin
      If Includes_check(s,t)
            Then return FAIL;
            Else return
                  Compose_substitutions(σ, {t/s})
End Procedure; {Variable_mgu}
```

```
Procedure Term_list_mgu
                      (s, t: list_of_terms; σ: substitution);
                                        Returns: substitution;
--- If there is an mgu τ for the lists s and t, returns
--- στ.   Returns FAIL otherwise.
   Begin
     If  s = ⟨⟩
        then return σ;
        else  return
          Term_list_mgu_aux(
                         Rest(s),
                         Rest(t),
                         Term_mgu(first(s), first(t), ∅)),
                         σ)
      End if;
   End Procedure; {Term_list_mgu}
```



```
Procedure Term_list_mgu_aux
                     (s, t: list_of_terms; τ, σ: substitution);
                                        Returns: substitution;
--- Auxiliary function to support Term_list_mgu.
  Begin
     Term_list_mgu(
          Apply_substitution_to_list(s, τ),
          Apply_substitution_to_list(t, τ),
          Compose_substitutions(σ,τ) )
  End Procedure; {Term_list_mgu_aux}
```

```
Procedure Apply_substitution_to_list
              (x: list_of_terms, σ: substitution);
--- Applies the substitution σ to every term in the list
--- x.
```

```
Procedure Atom_mgu
           (ψ1, ψ2: logical_atom);
                              Returns: substitution;
--- If  ψ1 and ψ2 have the same relation name,
--- and if the corresponding lists of terms are
--- unifiable, returns σ composed with the mgu
--- for those lists.
--- Returns FAIL otherwise.
 Begin
     If  Relation_name(ψ1) = Relation_name(ψ2)
       Then
         Term_list_mgu(arg_list(A), arg_list(B),  ∅);
      Else Return FAIL;
 End Procedure; {Atom_mgu}
```

- The overall algorithm is invoked with a call to Atom_mgu.
- Note that this algorithm is *tail recursive*. Once an instance of a procedure calls another procedure, the calling instance may be discarded.
- This implies that the entire algorithm may be implemented iteratively, without a deep stack.
- The sequence of calls for the running example is shown on the next slide.

The tail-recursive call graph for the processing of ψ_1 and ψ_2 is shown below. Only the most significant procedure calls are shown:

Atom_mgu(P(a,x,f(g(y))), P(z,f(z),f(w)))

↓

Term_list_mgu(⟨a, x, f(g(y))⟩, ⟨z, f(z), f(w))⟩, ∅)

↓

Term_list_mgu_aux(⟨x, f(g(y))⟩, ⟨f(z), f(w))⟩, Term_mgu(a, z, ∅), ∅)

↓ Variable_mgu(z, a, ∅) ↓ {a/z}

Term_list_mgu(⟨x, f(g(y))⟩, ⟨f(a), f(w))⟩, {a/z}

↓

Term_list_mgu_aux(⟨f(g(y))⟩, ⟨f(w))⟩, Term_mgu(x, f(a), ∅), {a/z})

↓ Variable_mgu(x, f(a), ∅) ↓ {f(a)/x}

Term_list_mgu(⟨⟩, ⟨⟩, Term_mgu(f(g(y)), f(w), {a/z, f(a)/x}))

↓

Term_list_mgu(⟨g(y)⟩, ⟨w⟩, {a/z, f(a)/x}))

↓

Term_list_mgu_aux(⟨⟩, ⟨⟩, Term_mgu(g(y), w, ∅) {a/z, f(a)/x}))

↓ Variable_mgu(w, g(y), ∅) ↓ { g(y)/w}

Term_list_mgu(⟨⟩, ⟨⟩, {a/z, f(a)/x, g(y)/w}))

↓

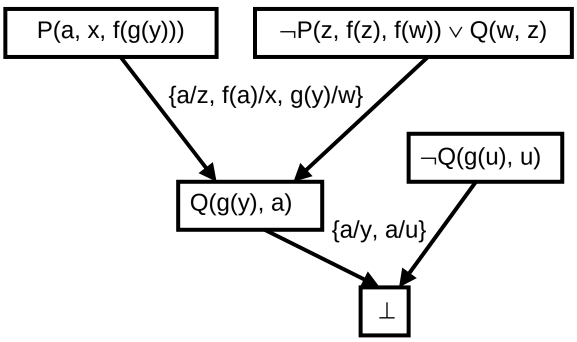
{a/z, f(a)/x, g(y)/w}

A simple resolution example:

Suppose that we are given the following clauses:

P(a, x, f(g(y)))
¬P(z, f(z), f(w)) ∨ Q(w, z)
¬Q(g(u), u)

Here is a resolution refutation:



- Note that the unifying substitutions are applied to entire clauses, and not just to the atoms to be matched.

Renaming of variables and re-use of clauses:

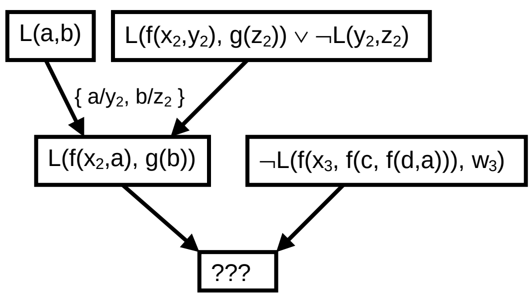
Consider the problem of showing that the following set of clauses is unsatisfiable.

Φ = {L(a,b),
L(f(x,y), g(z)) ∨ ¬L(y,z),
¬L(f(x, f(c, f(d,a))), w) }

Since the clauses contain variable names in common, the first step is to rename variables.

Φ' = {L(a,b),
L(f(x_2,y_2), g(z_2)) ∨ ¬L(y_2,z_2),
¬L(f(x_3, f(c, f(d,a))), w_3) }

Here is a first attempt at a refutation proof using resolution.



Is it possible to proceed and shown that the set is unsatisfiable?

Yes. To proceed, it is necessary to employ clause re-use.

- Notice that variable renaming "on the fly" is required to avoid collision of variable names.