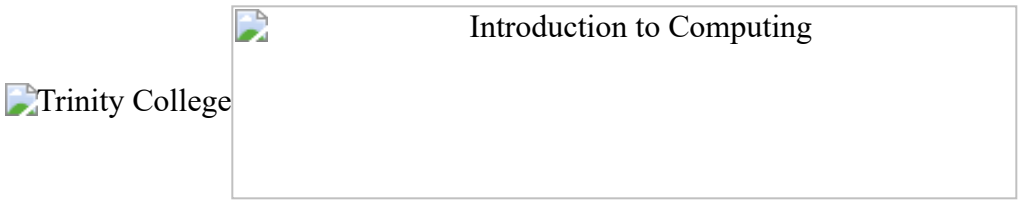

URL: http://www.cs.trincoll.edu/~ram/cpsc352/notes/unification.html
Last updated: 09/24/2011 01:31:09

CPSC 352 -- Artificial Intelligence
Notes: Unification

In order to apply the rules of inference, an inference system must be able to determine when two expressions ***match***. ***Unification*** is an algorithm for determining the substitutions needed to make two predicate calculus expressions match. The algorithm described here is essentially the one used in the PROLOG language, which we will begin studying next week.

Some Assumptions

In order to use unification and inference rules in an inference system (such as PROLOG), the sentences must be expressed in a suitable form. The following set of assumptions are based on PROLOG, where the Unification algorithm plays an important role.

- All variables must be universally quantified. Whenever you see a variable in an expression, assume that it is universally quantified. This allows us full freedom in making substitutions.
- Existentially quantified variables may be eliminated by replacing them with constants that make the sentence true.
  For example, in `∃X mother(X,bill)`, we can replace **X** with a constant designating bill's mother, **ann**, to get: mother(ann, bill).

Replacement in Unification Algorithm

It's important to remember that in the unification algorithm a variable can be replaced by any term, including other variables and function expressions or arbitrary complexity, and including expressions that themselves contain variables.

For example, the function expression, `father(jack)`, may be substituted for X in `human(X)` to give `human(father(jack))`.

For example, here are some valid substitution instances of `foo(X,a,goo(Y))`:

```
foo(fred,a,goo(Z))    where fred is substituted for X and Z for Y, i.e., { fred/X, Z/Y }
foo(W,a,goo(jack))    where { W/X, jack/Y }
foo(Z,a,goo(moo(Z)))  where { Z/X, moo(Z)/Y }
```

We use the notation **X/Y** to indicate that **X** is substituted for the variable **Y**. We also refer to these as ***bindings***. So **Y** is bound to **X**.

The Occurs Check

A variable cannot be unified with a term containing that variable. So **X** cannot be replaced by **p(X)**, because this would create an infinite regression: **p(p(p(...X)...)**.

Variable Bindings Persist

Within a given scope, once a variable is bound, it may not be given a new binding in future unifications and inferences.

Composition

Is **S** and **S'** are two substitution sets, then the composition of **S** and **S'**, **SS'**, is obtained by applying **S** to the elements of **S'** and adding the result to **S**.

For example, the three sets { X/Y, W/Z }, { V/X }, { a/V, f(b)/W } are together equivalent to { a/Y, f(b)/Z } because *a* can be substituted for *V*, which is then substituted for *X*, which is then substituted for *Y*, so *a* is substituted for *Y*. Similarly, *f(b)* is substituted for *W*, which is substituted for *Z*, which gives *f(b)/Z*.

Most General Unifier

If **s** is any unifier of a set of expressions **E**, and **g** is the most general unifier of that set of expressions, then for **s** applied to **E** there exists another unifier **s'** such that **Es = Egs'** (where **gs'** is the composition of unifiers).

In plain English, a most general unifier is just what its name implies: the most general set of substitutions that can be found for the two expressions.

Example: Suppose you have two expressions **p(X)** an **p(Y)**. One way to unify these is to substitute any constant expression for X and Y: { fred/X, fred/Y }. But this is not the most general unifier, because if we substitute any variable for X and Y, we get a more general unifier: { Z/X, Z/Y }. The first unifier is a valid unifier, but it would lessen the generality of inferences that we might want to make.

```
Let E = { p(X), p(Y) }
Let s = { fred/X, fred/Y }
Let g = { Z/X, Z/Y }
Now let s' = { fred/Z }

Then Es = { p(fred), p(fred) }
and gs' = { fred/X, fred/Y }
and therefore Egs' = { p(fred), p(fred) } = Es
```

So, given the most general unifier, you can always create a less general unifier by means of composition.

The Unification Algorithm

Unification performs *syntactic pattern matching*. Therefore the algorithm doesn't care about the difference between predicates, constants, variables, and functions. It only cares about symbols.

To simplify the algorithm we will therefore use list notation for our expressions, whereby a predicate name or function name is placed together with the predicate's or function's parameters in a single list:

```
CONVENTIONAL NOTATION     LIST NOTATION
---------------------     -------------
p(a,b)                    (p a b)
p(f(a),g(X,Y))            (p (f a) (g X Y))
equal(eve,mother(cain))   (equal eve (mother cain))
```

The Unify function

```
function unify(E1, E2);
  begin
    case
      both E1 and E2 are constants or the empty list:  % base case
        if E1 = E2
          then return {}
          else return FAIL;

      E1 is a variable:
        if E1 occurs in E2
          then return FAIL;
          else return {E2/E1}

      E2 is a variable:
        if E2 occurs in E1
          then return FAIL;
          else return {E1/E2}

      otherwise:                              % both E1 and E2 are lists
        begin
          HE1 := first element of E1;
          HE2 := first element of E2;
          SUBS1 := unify(HE1, HE2);            %   recursion
          if SUBS1 = FAIL, then return FAIL;
          TE1 := apply(SUBS1, rest of E1);
          TE2 := apply(SUBS1, rest of E2);
          SUBS2 := unify(TE1, TE2);            %  recursion
          if SUBS2 = FAIL then return FAIL;
             else return composition(SUBS1, SUBS2)
        end
  end
```

Example: Unify the expressions: (parents X (father X) (mother bill)) and (parents bill (father bill) Y).

Exercises Attempt to unify the following expressions. Either show their most general unifier or explain why they will not unify.

1. p(foo(X), Y) and p(a, b)
2. p(Y, Y) and p(a, Y)

Questions or Suggestions: ralph.morelli@trincoll.edu