Unification

1 Introduction

Pattern-matching —the ability to see that one description has the same structure as another— is obviously fundamental to intelligence. In this lecture, we look at one example of how to do pattern-matching. In this case, however, the patterns we are matching are expressions of logic. To be more precise, they're *atoms*.

So, in this lecture, we stop looking at logic from a mathematical point of view, and start to look at it from a computational point of view. (We will need to return to looking at it from a more mathematical point of view later in the course.) We're interested in how to implement a particular operation. This operation is called *unification*.

2 Bindings and Substitutions

A *binding* is a mapping of a variable to some other expression, and is written:

$$V \mapsto e$$

We say that variable V is *bound* to expression e. e may be a constant symbol, another variable, or a function symbol with its arguments (in other words, any *term*). However, there is another restriction: e cannot contain V, e.g. $x \mapsto f(x)$ is illegal.

A *substitution* is a *set of bindings*, e.g. $\{x \mapsto a, y \mapsto z, z \mapsto f(a,b)\}$

There are two important operations (which we will use in our implementation of unification) that we must study at this point.

2.1 Applying a substitution

To *apply* a substitution to an expression is simply to replace variables in the expression by what they are bound to in the substitution. (We call this *instantiating* the variables.)

Fill in these examples in the lecture. Let e be $p(x, f(y), b)$

θ	$e\theta$
$\{x \mapsto z, y \mapsto w\}$	
$\{y \mapsto c\}$	
$\{x \mapsto g(z), y \mapsto d\}$	
$\{x \mapsto c, y \mapsto d\}$	

2.2 Composing two substitutions

To *compose* two substitutions is to bring the two sets of bindings together into a single set. But it's an operation that requires some care.

First, you *apply* the second set to each expression in the first set. Then, you add to the result any bindings from the second set whose variables do not appear among the variables of the first set.

Again, fill in these examples during the lecture.

θ	$\{z \mapsto f(x,y)\}$
τ	$\{x \mapsto b, y \mapsto c, w \mapsto d, z \mapsto e\}$
$\theta\tau$	

θ	$\{x_1 \mapsto f(y_1), x_2 \mapsto y_2, x_3 \mapsto g(y_1, y_2)\}$
τ	$\{y_1 \mapsto a, y_2 \mapsto y_3\}$
$\theta\tau$	

3 Unification

Two expressions, e_1 and e_2, *unify* if there exists a substitution θ such that

$$e_1\theta = e_2\theta$$

i.e. a substitution that, when applied to both expressions, makes them equal. We say that θ is a unifier of e_1 and e_2.

For example, these two expressions unify. (Fill in the unifier during the lecture)

$$p(a, x, f(y), b)$$

$$p(z, z, f(c), b)$$

But, what we generally want to come up with is the substitution that is the *most general unifier* (mgu) of the two expressions. For example, here's one unifier of these two expressions:

$$p(x, f(y), b)\,\{x \mapsto c, y \mapsto b\} \quad =$$

$$p(x, f(b), b)\,\{x \mapsto c, y \mapsto b\} \quad =$$

But here's another:

$$p(x, f(y), b)\,\{y \mapsto b\} \quad =$$

$$p(x, f(b), b)\,\{y \mapsto b\} \quad =$$

The second unifier, $\{y \mapsto b\}$, is more general than the first, $\{x \mapsto c, y \mapsto b\}$. It contains fewer bindings, and yet still makes the two expressions equal. (It has no unnecessary binding for x.)

We will want to compute the *most* general unifier: the one that makes fewest bindings (no unnecessary ones). If the expressions unify at all, there will be a unique most general unifier (up to variable renaming).

4 A Unification Algorithm

This algorithm computes the mgu of two atoms (if there is one). Remember, an atom is a predicate symbol with its arguments (each of which is a term).

- If the predicate symbols are not the same, or if the arities are not the same, return fail
- Otherwise, step thro' the two atoms, argument by argument
  - If the arguments are equal, go on to the next argument
  - If they are different,
    * If at least one is a variable,
      · create a binding (but do the *occurs check*!)
      · *apply* the binding to the rest of the atom
      · *compose* the binding with the set of bindings found so far
    * If not, but both are compound terms using the same function symbol,
      · step thro' the arguments of the function symbols
    * Otherwise, return fail

(The 'occurs check' is the check that we aren't forming an illegal binding such as $x \mapsto f(x)$.)

In lectures, we'll apply the algorithm to the following examples. Be very clear which symbols are constants and which are variables!

1) $p(a, b, c)$
   $q(a, b, c)$

2) $p(a, b, c, z)$
   $p(a, b, x, y)$

3) $p(a, b, c)$
   $p(a, x, x)$

4) $p(a, f(c, x), g(d, e))$
   $p(a, y, g(z, e))$

5) $p(a, f(b, z))$
   $p(a, g(b, c))$

6) $p(a, f(c, x), x)$
   $p(a, y, y)$

7) $p(a, f(x), x)$
   $p(a, y, c)$

Exercises

Determine whether the members of the following pairs of atoms unify with each other. If they do, give the mgu; if not, give a brief explanation.

1.	*colour*(*tweety*, *yellow*)	*colour*(x, y)
2.	*colour*(*tweety*, *yellow*)	*colour*(x, x)
3.	*colour*(*hat*(*postmanPat*), *blue*)	*colour*(*hat*$(y), x)$
4.	$r(f(x), b)$	$r(y, z)$
5.	$r(f(y), x)$	$r(x, f(b))$
6.	$r(f(y), y, x)$	$r(x, f(a), f(v))$
7.	*loves*(x, y)	*loves*(y, x)