Lecture 7 (Sep 25)

Lecture outline:

- examples of unification algorithm
- resolution for predicate logic
- example of refutation search tree

Examples for Robinson's unification algorithm presented in the last lecture:

Example 1. $U_1 = \{p(f(a), g(X)), p(Y, Y)\}$

θ	t_1	t_2	new subst.
identity	$p(f(a), g(X))$	$p(Y, Y)$	$[Y/f(a)]$
$[Y/f(a)]$	$p(f(a), g(X))$	$p(f(a), f(a))$	non-unifiable

Example 2. $U_2 = \{p(a, X, h(g(Z))), p(Z, h(Y), h(Y))\}$

θ	t_1	t_2	new subst.
identity	$p(a, X, h(g(Z)))$	$p(Z, h(Y), h(Y))$	$[Z/a]$
$[Z/a]$	$p(a, X, h(g(a)))$	$p(a, h(Y), h(Y))$	$[X/h(Y)]$
$[Z/a] \circ [X/h(Y)]$	$p(a, h(Y), h(g(a)))$	$p(a, h(Y), h(Y))$	$[Y/g(a)]$
$[Z/a] \circ [X/h(Y)] \circ [Y/g(a)]$	$p(a, h(g(a)), h(g(Z)))$	$p(a, h(g(a)), h(g(a)))$	done

The terms are unifiable, and the m.g.u. is

$$[Z/a] \circ [X/h(Y)] \circ [Y/g(a)] = [Z/a, X/h(g(a)), Y/g(a)]$$

Example 3. $U_3 = \{f(X, 2), f(h(X), 2)\}$

θ	t_1	t_2	new subst.
identity	$f(X, 2)$	$f(h(X), 2)$	non-unifiable (occurs check fails)

Resolution for Predicate Logic

Definition 4. *The* resolution rule *(for predicate logic) is an logical inference rule defined as follows: given two clauses* $C_1 = \alpha_1, t_1, \alpha_2 :- \alpha_3$ *and* $C_2 = \beta_1 :- \beta_2, t_2, \beta_3$*, where*

1. α_i *and* β_i *are arbitrary, possibly empty, sequences of literals;*
2. C_1 *and* C_2 *have no variables in common*
3. *Terms* t_1 *and* t_2 *are unifiable, with* θ *being their m.g.u.,*

derive a clause

$$\theta(\alpha_1), \theta(\alpha_2), \theta(\beta_1) :- \theta(\alpha_3), \theta(\beta_2), \theta(\beta_3).$$

Graphically, we picture an application of the resolution rule as follows:

$$\alpha_1, t_1, \alpha_2 :\!-\ \alpha_3 \qquad \beta_1 :\!-\ \beta_2, t_2, \beta_3$$

theta

$$\theta(\alpha_1), \theta(\alpha_2), \theta(\beta_1) :\!-\ \theta(\alpha_3), \theta(\beta_2), \theta(\beta_3)$$

Example 5.

$$p(X, f(2)) :\!-\ s(X, Z) \qquad r(T, g(Y)) :\!-\ p(g(T), f(Y))$$

[X/g(T),Y/2]

$$r(T, g(2)) :\!-\ s(g(T), Z)$$

Note that condition 2 in the Definition 4 is motivated by the following example:

$$p(X) :\!-$$
$$:\!-\ p(f(X))$$

Since the first clause says $\forall X p(X)$, and the second $\forall X \neg p(f(X))$, the two clauses are contradictory, however we cannot unify $p(X)$ and $p(f(X))$ because of the occurs check. Since the names of variables within a clause make no difference, in order to comply to condition 2, we simply rename the variable X by something else, like X' in one of the clauses. Then the two clauses become

$$p(X) :\!-$$
$$:\!-\ p(f(X'))$$

and now we can resolve them.

Theorem 6 (Robinson)**.** *A set S of predicate logic clauses is inconsistent if and only if* $:\!-$ *can be derived from S using the resolution rule.*

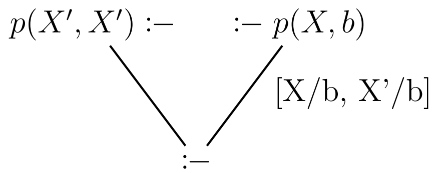
Example 7. Consider the following logic program:

$$C_1 : p(X, Z) :\!-\ q(X, Y), p(Y, Z)$$
$$C_2 : p(X, X) :\!-$$
$$C_3 : q(a, b) :\!-$$

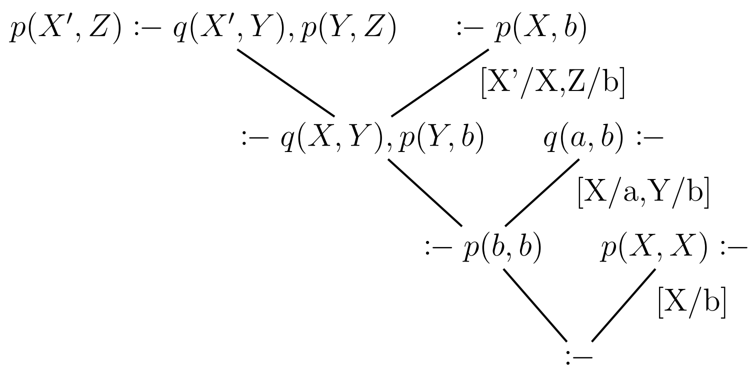
and the following goal clause:

$$:\!-\ p(X, b)$$

We will construct linear resolution refutation of these clauses. So, we start from the goal clause. Note that immediately we have a choice: we can resolve the goal with either C_1 or C_2. If we resolve with C_2 we obtain:



If instead we resolve with C_1 we obtain:



Note that after the first resolution, we also had a choice – this time whether to try to resolve the first term, or the second. And, also after the second resolution (when the goal was $p(b, b)$), we again could've chosen to resolve with C_1.

So, during the construction of linear resolution refutation we have the following choices:

1. If the goal clauses contains multiple terms, which of the terms to try first for resolution.

2. If there are more than one program clause which unifies with the term selected in 1., which of the program clauses to try first.

In standard Prolog, these choices are resolved in the following manner:

1. Always select the first (i.e. left-most) term in the current goal clause.

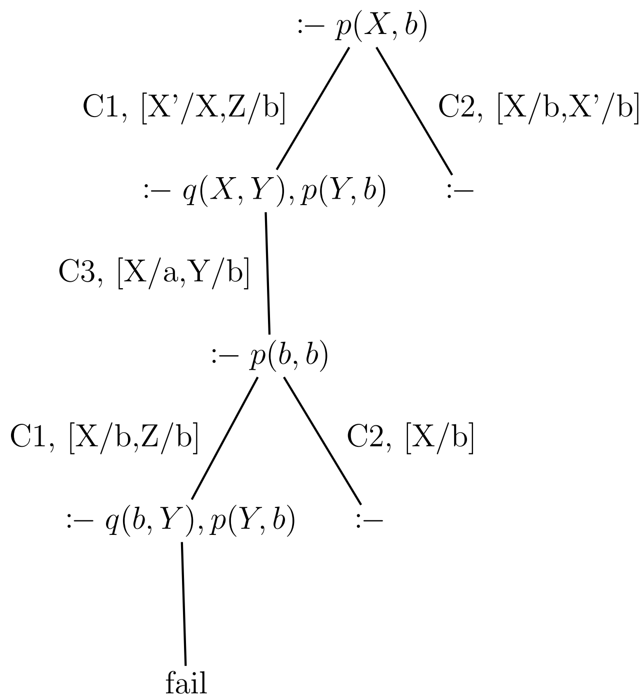
2. Try the program clauses in the "natural" order, that is, in the order in which the clauses are listed in the program.

Notice also, that by "collecting" unifiers from the path that leads to :– we can also obtain the answer to the original goal clause – that is values of any variables that appear in that clause. The answer is obtained by composing all unifiers on the path that leads from the goal to the empty clause, in the order of appearance. For the Example 7:

- In the first refutation, the composition of all unifiers is simply $[X/b, X'/b]$, and so the answer in this case is $X = b$.

- In the second refutation, the composition of all unifiers is $[X'/X, Z/b] \circ [X/a, Y/b] \circ [X/b] = [X'/a, X/a, Z/b, Y/b]$, and so the answer in this case is $X = a$.

This is exactly the mechanism the Prolog uses to answer our queries, as in the flight connection example we saw on the second lecture.
And, finally, the process of the search for resolution refutation can be captured nicely using a resolution search tree. In our example, the tree will look as follows:



The tree represents all possible ways to derive an empty clause using resolution. The "fail" means that we cannot resolve the first term of the current goal with any of the clauses in the program. As with the examples above, whenever an empty clause is derived, the answer to the original goal is obtained by composition of all the unifiers that lead from the root of the tree to the empty clause. Thus, for the path on the right the answer will be $X = a$, because

$$[X'/X, Z/b] \circ [X/a, Y/b] \circ [X/b] = [X'/a, X/a, Z/b, Y/b]$$

, and for the path on the right, the answer will be $X = b$.