Lecture 6 (Sep 23)

Lecture outline:

- example for conversion of predicate logic formulas to CNF
- substitutions
- unification
- Robinson's unification algorithm

In the last lecture, we started to talk about the conversion of predicate formulas to logic programing clauses. The first two steps of the conversion are

1. Move all quantifiers outside. This gives a formula in prenex normal form.
2. Eliminate the existential quantifiers using Skolemization.

After these steps the formula is in prenex normal form with only universal quantifiers, so the last step is

3. Drop the universal quantifiers, and convert the resulting formula to logic programming clauses, as if it was a propositional formula.

Example 1. (From the Sep 18 exercise) Convert the following formula to logic programming clauses

$$(\exists X \forall Y p(X,Y)) \vee \neg \exists Y (q(Y) \rightarrow \forall Z r(Z))$$

Ok, there we go:

$$\begin{aligned}
&(\exists X \forall Y p(X,Y)) \vee \neg \exists Y (q(Y) \rightarrow \forall Z r(Z)) && \\
&(\exists X \forall Y p(X,Y)) \vee \neg \exists Y (\neg q(Y) \vee \forall Z r(Z)) && \text{(replaced } \rightarrow\text{)} \\
&(\exists X \forall Y p(X,Y)) \vee \forall Y (q(Y) \wedge \exists Z \neg r(Z)) && \text{(moved } \neg \text{ inwards)} \\
&\exists X [(\forall Y p(X,Y)) \vee \forall Y (q(Y) \wedge \exists Z \neg r(Z))] && \text{(moved } \exists X \text{ out)} \\
&\exists X \forall T [p(X,T) \vee \forall Y (q(Y) \wedge \exists Z \neg r(Z))] && \text{(moved the first } \forall Y\text{, had to rename } Y \text{ with } T\text{)} \\
&\exists X \forall T \forall Y [p(X,T) \vee (q(Y) \wedge \exists Z \neg r(Z))] && \text{(moved } \forall Y\text{)} \\
&\exists X \forall T \forall Y \exists Z [p(X,T) \vee (q(Y) \wedge \neg r(Z))] && \text{(moved } \exists Z\text{)} \\
&\forall T \forall Y \exists Z [p(c,T) \vee (q(Y) \wedge \neg r(Z))] && \text{(eliminated } \exists X\text{)} \\
&\forall T \forall Y [p(c,T) \vee (q(Y) \wedge \neg r(f(T,Y)))] && \text{(eliminated } \exists Z\text{)} \\
&p(c,T) \vee (q(Y) \wedge \neg r(f(T,Y))) && \text{(dropped all universal quantifiers)} \\
&(p(c,T) \vee q(Y)) \wedge (p(c,T) \vee \neg r(f(T,Y))) && \text{(distributive law)}
\end{aligned}$$

Now we have CNF, which gives us the following two logic programming clauses:

$$\begin{aligned}
p(c,T), q(Y) &:- \\
p(c,T) &:- r(f(T,Y))
\end{aligned}$$

Unification

Unification is an extra step needed for resolution for predicate logic. Take for example two formulas $\forall X f(X)$ and $\forall Y \neg f(Y)$, which give two logic programming clauses $f(X) :-$ and $:- f(Y)$. Using propositional resolution rule we cannot resolve these, but, intuitively we should be able to, because we can just replace X by Y in the first clause. Intuitively, unification is the process of "making terms look the same" so the resolution rule can be applied. In order to formalize this notion we need a few concepts.

Definition 2. *A* substitution *is a function from variables to terms. Substitutions are denoted in the following way: if a substitution θ maps variables $V_1, \ldots, V_n$ to terms $t_1, \ldots, t_n$, we write*

$$\theta = [V_1/t_1, \ldots, V_n/t_n]$$

The result of the application of the substitution θ to a term t*, denoted as $\theta(t)$, is a term obtained by the simultaneous replacement of each occurrence in t of a variable from $Dom(\theta)$ with a corresponding term from $Range(\theta)$.*

Example 3. Let $\theta_1 = [X/2, Y/h(Z)]$, and $t = f(X, g(Y, a))$, then

$$\theta_1(t) = f(2, g(h(Z), a)).$$

For $\theta_2 = [X/2, Y/h(X)]$, we have

$$\theta_2(t) = f(2, g(h(X), a)),$$

and **not** $f(2, g(h(2), a))$.

Definition 4. *Let θ and δ be two substitutions. The* composition *of θ and δ, denoted as $\theta \circ \delta$, is a substitution which maps every term t to a term $\delta(\theta(t))$.*

Example 5. Let $t = g(X, f(T))$, and

$$\theta = [X/f(Y), Y/g(2, Z)]$$
$$\delta = [Y/h(2), Z/3, T/h(a, Z)]$$

Then,

$$\theta \circ \delta(t) =$$
$$\delta(\theta(t)) =$$
$$\delta(g(f(Y), f(T)) =$$
$$g(f(h(2)), f(h(a, Z))),$$

and

$$\theta \circ \delta = [X/h(2), Y/g(2, 3), Z/3, T/h(a, Z)]$$

Definition 6. *Let $U = \{t_1, t_2, \ldots, t_n\}$ be a set of terms. A substitution θ is called a unifier of U if*

$$\theta(t_1) = \theta(t_2) = \cdots = \theta(t_n)$$

If U has a unifier, U is called unifiable*, otherwise* not unifiable*.*

Example 7. The set $U_1 = \{f(X, 2), f(h(Y), 2)\}$ is unifiable - the substitution $\theta = [X/h(Y)]$ is a unifier for U_1.

Example 8. The set $U_2 = \{f(X, 2), f(h(Y), Y)\}$ is unifiable - the substitution $\theta = [X/h(2), Y/2]$ is a unifier for U_2.

Example 9. The set $U_3 = \{f(X, 2), f(h(X), 2)\}$ is not unifiable.

Note that in the Example 7 we could have chosen a different unifier. For example, $\theta' = [X/h(23), Y/23]$ is also a unifier for U_1, and so are infinitely many other ones. Intuitively, we want a "simplest" possible unifier, i.e. the one that does the smallest amount of changes. This is captured by the following concept:

Definition 10. *A unifier θ of a set of terms U is* most general unifier (m.g.u) *if and only if for every unifier ε of U there exists a substitution δ such that $\varepsilon = \theta \circ \delta$.*

Example 11. The unifier $\theta = [X/h(Y)]$ in Example 7 is m.g.u. of the set $U_1 = \{f(X, 2), f(h(Y), 2)\}$. For example, a unifier $\theta' = [X/h(23), Y/23]$ can be obtained from θ using the substitution $[Y/23]$:

$$[X/h(23), Y/23] = [X/h(Y)] \circ [Y/23]$$

Intuition[1]: an m.g.u. is sort-of a "generator" for all other unifiers – every other unifier can be obtained from it via some substitution.

Unification Algorithm

The algorithm presented below is due to Alan Robinson, and allows to find an m.g.u. of two terms, if they are unifiable. Note that there is a slightly more complicated but more efficient version of unification algorithm, used in Prolog.
The condition checked on line 10 is called the *occurs check*. It is designed to handle cases similar to what we saw in Example 9.
Note that the operator = in Prolog stands for "unifiable", i.e. in Prolog $t_1 = t_2$ is true if and only if t_1 and t_2 are unifiable. Note also, that Prolog omits the occurs check, and so, for example, $X = f(X)$ is true in Prolog.

[1] I think now it does makes sense

Algorithm 1 UNIFY([*in*] t_1, [*in*] t_2)

Input: t_1, t_2 – two terms.
Output: An m.g.u. of t_1 and t_2 if the set $\{t_1, t_2\}$ is unifiable; NOT-UNIFIABLE otherwise.

1: $\theta \leftarrow$ identity substitution (i.e. $\theta(V) = V$ for all V)
2: $p_1 \leftarrow$ pointer to the first symbol of t_1
3: $p_2 \leftarrow$ pointer to the first symbol of t_2
4: **while** $t_1 \neq t_2$ **do**
5: Advance p_1 and p_2 *simultaneously* until a pair of different symbols is found
6: **if** neither p_1 nor p_2 point to a variable **then**
7: **return** NON-UNIFIABLE
8: **end if**
9: Let X be the variable, and t be the term (may be a variable) pointed by p_1 and p_2
10: **if** X occurs in t **then** ▷ occurs check
11: **return** NON-UNIFIABLE
12: **end if**
13: $t_1 = [X/t](t_1)$
14: $t_2 = [X/t](t_2)$
15: $\theta = \theta \circ [X/t]$
16: **end while**
17: **return** θ