COMP30026 Models of Computation

Lecture 8: Predicate Logic: Unification and Resolution

Mak Nazecic-Andrlon and William Umboh

Semester 2, 2024

Notation for Variables and Constants

Recall again our convention:

- Letters from the start of the alphabet (a, b, c, $\dots$) are for constants.
- Letters from the end of the alphabet (u, v, x, y, $\dots$) are for variables.

This distinction is very important in what follows.

Substitutions

A substitution is a function from variables to terms.

Notation: $\theta = \{x_1 \mapsto t_1, x_2 \mapsto t_2, \ldots, x_n \mapsto t_n\}$.

Given expression (i.e. formula or term) E, denote by $E[\theta]$ the result of simultaneously replacing each occurrence of x_i in E by t_i.

Example: If E is $P(f(x), g(y, y, b))$, and

$$\theta = \{x \mapsto h(u), y \mapsto a, z \mapsto c\}$$

then $E[\theta]$ is $P(f(h(u)), g(a, a, b)\}$.

Unifiers

A unifier of two expressions s and t is a substitution θ such that $s[\theta] = t[\theta]$.

s and t are unifiable iff there exists a unifier for s and t.

Example

$L(x)$ and $L(c)$ are unifiable:

Both $L(x)[\theta]$ and $L(c)[\theta]$ are $L(c)$, when $\theta = \{x \mapsto c\}$.

Question: Are $L(a)$ and $L(c)$ unifiable?

Most General Unifiers

A most general unifier (mgu) for s and t is a substitution θ such that

1. θ is a unifier for s and t, and
2. every other unifier σ of s and t can be expressed as $\tau \circ \theta$ for some substitution τ.

(The composition $\tau \circ \theta$ is the substitution which first applies θ, and then applies τ to that result.)

Theorem. If s and t are unifiable, they have a most general unifier.

Unifier Examples

1. $P(x, a)$ and $P(b, c)$ are not unifiable.
2. $P(f(x), y)$ and $P(a, w)$ are not unifiable.
3. $P(x, c)$ and $P(a, y)$ are unifiable using $\{x \mapsto a, y \mapsto c\}$.
4. $P(f(x), c)$ and $P(f(a), y)$ also unifiable using $\{x \mapsto a, y \mapsto c\}$.
5. **Note:** $P(x)$ and $P(f(x))$ are not unifiable.

If we were allowed to have a substitution $\{x \mapsto f(f(f(\ldots)))\}$, that would be a unifier for the last example. But we cannot have that, as terms must be finite.

More Unifier Examples

Now consider $P(f(x), g(y, a))$ and $P(f(a), g(z, a))$.

The following are all unifiers, so which is "best"?

- $A = \{x \mapsto a, y \mapsto z\}$
- $B = \{x \mapsto a, y \mapsto a, z \mapsto a\}$
- $C = \{x \mapsto a, y \mapsto g(b, f(u)), z \mapsto g(b, f(u))\}$
- $D = \{x \mapsto a, z \mapsto y\}$

A and D are mgus. They avoid making unnecessary commitments.

B needlessly commits y and z to be a.

Note that $B = \{y \mapsto a\} \circ D$.

A Syntactic Unification Algorithm

Input: Two expressions s and t.

Output: If they are unifiable: a most general unifier for s and t; otherwise 'failure'.

Algorithm:

1. Start with this set consisting of one equation: $\{s = t\}$.
2. As long as some equation in the set has one of the six forms listed on the next slide, perform the corresponding action.
3. Return the result.

Unification: Solving Term Equations

In the following, let x be a variable and let F and G be function or predicate symbols.

1. $F(s_1, \ldots, s_n) = F(t_1, \ldots, t_n)$:
   - Replace the equation by the n equations $s_1 = t_1, \ldots, s_n = t_n$.
2. $F(s_1, \ldots, s_n) = G(t_1, \ldots, t_m)$ where $F \neq G$ or $n \neq m$:
   - Halt, returning 'failure'.
3. $x = x$:
   - Delete the equation.
4. $t = x$ where t is not a variable:
   - Replace the equation by $x = t$.
5. $x = t$ where t is not x but x occurs in t:
   - Halt, returning 'failure'.
6. $x = t$ where t contains no x but x occurs in other equations:
   - Replace x by t in those other equations.

Solving Term Equations: Example 1

Starting from

$$f(h(y), g(y, a), z) = f(x, g(v, v), b)$$

we rewrite:

$$\stackrel{(1,4)}{\Longrightarrow} \begin{array}{rcl} x & = & h(y) \\ g(y, a) & = & g(v, v) \\ z & = & b \end{array} \stackrel{(1,4)}{\Longrightarrow} \begin{array}{rcl} x & = & h(y) \\ y & = & v \\ v & = & a \\ z & = & b \end{array} \stackrel{(6,6)}{\Longrightarrow} \begin{array}{rcl} x & = & h(a) \\ y & = & a \\ v & = & a \\ z & = & b \end{array}$$

The last set is in normal form and corresponds to the substitution

$$\{x \mapsto h(a), y \mapsto a, v \mapsto a, z \mapsto b\}$$

which indeed unifies the two original terms.

Solving Term Equations: Example 2

Starting from

$$f(x, a, x) = f(h(z, b), y, h(z, y))$$

we rewrite:

$$\stackrel{(1,4)}{\Longrightarrow} \begin{array}{rcl} x & = & h(z,b) \\ y & = & a \\ x & = & h(z,y) \end{array} \stackrel{(6)}{\Longrightarrow} \begin{array}{rcl} x & = & h(z,b) \\ y & = & a \\ h(z,b) & = & h(z,y) \end{array} \stackrel{(1,4)}{\Longrightarrow} \begin{array}{rcl} x & = & h(z,b) \\ y & = & a \\ z & = & z \\ y & = & b \end{array}$$

$$\stackrel{(3)}{\Longrightarrow} \begin{array}{rcl} x & = & h(z,b) \\ y & = & a \\ y & = & b \end{array} \stackrel{(6)}{\Longrightarrow} \begin{array}{rcl} x & = & h(z,b) \\ y & = & a \\ a & = & b \end{array} \stackrel{(2)}{\Longrightarrow} \text{failure}$$

So the two original terms are not unifiable.

Solving Term Equations: Example 3

Starting from

$$f(x, g(v, v), x) = f(h(y), g(y, z), z)$$

we rewrite:

$$\xRightarrow{(1)} \begin{array}{rcl} x & = & h(y) \\ g(v, v) & = & g(y, z) \\ x & = & z \end{array} \xRightarrow{(6,4)} \begin{array}{rcl} x & = & h(y) \\ g(v, v) & = & g(y, z) \\ z & = & h(y) \end{array}$$

$$\xRightarrow{(1)} \begin{array}{rcl} x & = & h(y) \\ v & = & y \\ v & = & z \\ z & = & h(y) \end{array} \xRightarrow{(6)} \begin{array}{rcl} x & = & h(y) \\ z & = & y \\ v & = & z \\ z & = & h(y) \end{array} \xRightarrow{(6)} \begin{array}{rcl} x & = & h(y) \\ z & = & y \\ v & = & y \\ y & = & h(y) \end{array} \xRightarrow{(5)} \text{failure}$$

This is "failure by occurs check": The algorithm fails as soon as we discover the equation $y = h(y)$.

Term Equations as Substitutions

This algorithm always halts.

If the result is 'failure', no unifier exists.

Otherwise, the term equation system is in normal form:

- Every LHS is a different variable.
- No LHS appears in any RHS.

If the normal form is $\{x_1 = t_1, \ldots, x_n = t_n\}$ then

$$\{x_1 \mapsto t_1, \ldots, x_n \mapsto t_n\}$$

is a mgu for the input terms.

Resolution for Predicate Logic

Let P_1 and P_2 be unifiable atomic formulas with no variables in common. Let θ be the unifier.

Let C_1 and C_2 be disjunctive clauses.

$$\frac{\begin{array}{c} C_1 \cup \{P_1\} \\ C_2 \cup \{\neg P_2\} \end{array}}{(C_1 \cup C_2)[\theta]}$$

Automated Inference with Predicate Logic

- Every shark eats a tadpole

$$\forall x(S(x) \rightarrow \exists y(T(y) \land E(x, y)))$$

- All large white fish are sharks

$$\forall x(W(x) \rightarrow S(x))$$

- Camilla is a large white fish living in deep water

$$W(camilla) \land D(camilla)$$

- Any tadpole eaten by a deep water fish is miserable

$$\forall z((T(z) \land \exists y(D(y) \land E(y, z))) \rightarrow M(z))$$

- Therefore some tadpole is miserable

$$\therefore \exists z(T(z) \land M(z))$$

Tadpoles in Clausal Form

- Every shark eats a tadpole

$$\{\neg S(x), T(f(x))\}, \{\neg S(x), E(x, f(x))\}$$

- All large white fish are sharks

$$\{\neg W(x), S(x)\}$$

- Camilla is a large white fish living in deep water

$$\{W(camilla)\}, \{D(camilla)\}$$

- Any tadpole eaten by a deep water fish is miserable

$$\{\neg T(z), \neg D(y), \neg E(y, z), M(z)\}$$

- Negation of: Some tadpole is miserable.

$$\{\neg T(z), \neg M(z)\}$$

A Refutation

Let us find a refutation of the set of seven clauses.

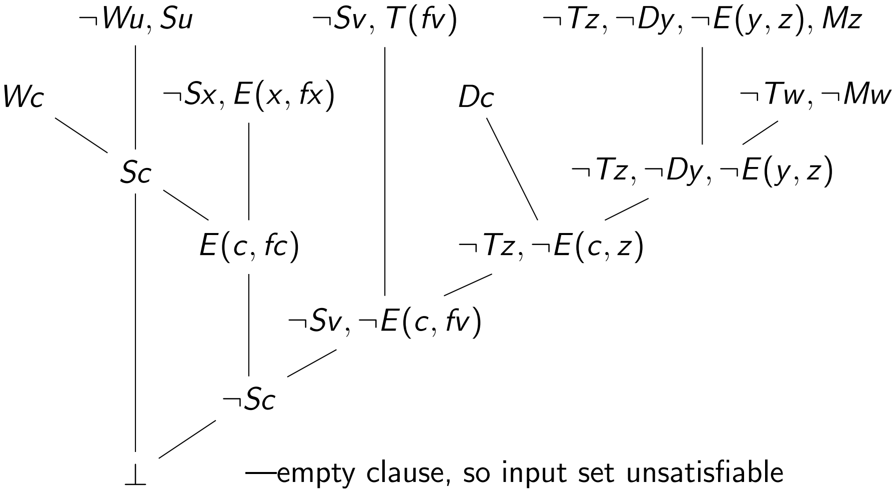
To save space, we leave out braces and some parentheses, for example, we write $\neg Wu, Su$ for clause $\{\neg W(u), S(u)\}$.

$$\neg Wu, Su \qquad \neg Sv, T(fv) \qquad \neg Tz, \neg Dy, \neg E(y,z), Mz$$

$$Wc \qquad \neg Sx, E(x, fx) \qquad Dc \qquad \neg Tw, \neg Mw$$

Many different resolution proofs are possible—the next slides show one.

A Refutation for the Tadpole Example



Resolution Exercise

Using resolution, justify this argument:

- All philosophers are wise $\forall x(P(x) \rightarrow W(x))$
- Some Greeks are philosophers $\exists x(G(x) \land P(x))$
- Therefore some Greeks are wise $\exists x(G(x) \land W(x))$

Factoring

In addition to resolution, there is one more valid rewriting of clauses, called factoring.

Let C be a clause and let $A_1, A_2 \in C$. If A_1 and A_2 are unifiable with mgu θ, add the clause $C[\theta]$.

$$\{D(f(y), y), \neg P(x), D(x, y), \neg P(z)\}$$

$$\{D(f(y), y), \neg P(f(y)), \neg P(z)\}$$

$$\{D(f(y), y), \neg P(f(y))\}$$

The Need for Factoring

Factoring is sometimes crucial:

$$
\begin{tikzcd}
\{P(x), P(y)\} \arrow[d, dash] & & \{\neg P(u), \neg P(v)\} \arrow[d, dash] \\
\{P(x)\} \arrow[dr, dash] & & \{\neg P(u)\} \arrow[dl, dash] \\
& \bot &
\end{tikzcd}
$$

How to Use Clauses

A resolution step uses two clauses (or two "copies" of the same clause). A factoring step uses one clause.

A given clause can be used many times in a refutation, taking part in many different resolution/factoring steps.

But recall that each clause is implicitly universally quantified.

Hence we really should rename all variables in a clause every time we use the clause, using fresh variable names.

Sometimes this renaming is essential for correctness, especially when resolution uses two "copies" of the same clause.

The Resolution Method

Start with collection $\mathcal{C}$ of clauses
While $\perp \notin \mathcal{C}$ do
  add to $\mathcal{C}$ a factor of some $C \in \mathcal{C}$
  or a resolvent of some $C_1, C_2 \in \mathcal{C}$

Question: Does this process always terminate for unsatisfiable inputs? For satisfiable inputs?

The Need for Fairness

For unsatisfiable inputs, only if we use a sensible search strategy. Consider:

$$\{Q\} \qquad \{\neg Q\} \qquad \{\neg P(x), P(f(x))\}$$

$$*$$

$$\{\neg P(x), P(f(f(x)))\}$$

$$\{\neg P(x), P(f(f(f(x))))\}$$

$$\vdots$$

***** Here we resolve the clause against a **renamed** copy of itself. Without renaming, unification fails.

The Power of Resolution

Theorem. $\mathcal{C}$ is unsatisfiable iff the resolution method can add $\perp$ after a finite number of steps.

We say that resolution is sound and refutation-complete.

Not a decision procedure: may not terminate on satisfiable formulas.

Indeed, no such procedure can exist for predicate logic.

Validity/unsatisfiability are semi-decidable properties.