COMP30026 Models of Computation

Predicate Logic: Unification and Resolution

Bach Le / Anna Kalenkova

Lecture Week 5 Part 1 (Zoom)

Semester 2, 2021

This Lecture is Being Recorded

Notation for Variables and Constants

Recall again our convention: We use letters from the start of the alphabet (a, b, c, ...) for constants, and letters from the end of the alphabet (u, v, x, y ...) for variables.

This distinction is very important in what follows.

We also usually use lower case letters such as f, g, h, ... as function symbols, and, of course, upper case letters for predicate symbols.

In some contexts it is important to distinguish function symbols and predicate symbols. As far as unification is concerned, however, there is no difference—the unification algorithm treats $f(x, a)$ and $P(f(x, a), x)$ the same way, so for now, let us just consider both "terms".

Substitutions

A substitution is a finite set of replacements of variables by terms, that is, a set θ of the form $\{x_1 \mapsto t_1, x_2 \mapsto t_2, \ldots, x_n \mapsto t_n\}$, where the x_i are variables and the t_i are terms.

We can also think of θ as a function from terms to terms, or from atomic formulas to atomic formulas. $\theta(F)$ is the result of simultaneously replacing each occurrence of x_i in F by t_i.

Example. If F is $P(f(x), g(y, y, b))$ and $\theta = \{x \mapsto h(u), y \mapsto a, z \mapsto c\}$ then $\theta(F)$ is $P(f(h(u)), g(a, a, b)\}$.

Note: Similar to a valuation, but a substitution maps a variable to a term, and, by natural extension, terms to terms.

Most General Unifiers

A unifier of two terms s and t is a substitution θ such that $\theta(s) = \theta(t)$.

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

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 we get by first using θ, and then using τ on the result produced by θ.)

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.

The last case relies on a principle that a variable (such as x) is not unifiable with any term containing x (apart from x itself).

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"?

- $\{x \mapsto a, y \mapsto z\}$
- $\{x \mapsto a, y \mapsto a, z \mapsto a\}$
- $\{x \mapsto a, y \mapsto g(b, f(u)), z \mapsto g(b, f(u))\}$
- $\{x \mapsto a, z \mapsto y\}$

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"?

- $\{x \mapsto a, y \mapsto z\}$
- $\{x \mapsto a, y \mapsto a, z \mapsto a\}$
- $\{x \mapsto a, y \mapsto g(b, f(u)), z \mapsto g(b, f(u))\}$
- $\{x \mapsto a, z \mapsto y\}$

The first and the last are mgus. They avoid making unnecessary commitments. The second commits y and z to take the value a, which was not really needed in order to unify the two formulas.

Note that $\{x \mapsto a, y \mapsto a, z \mapsto a\} = \{z \mapsto a\} \circ \{x \mapsto a, y \mapsto z\}$.

A Unification Algorithm

In the following, let x be a variable, let F and G be function or predicate names, and let s and t be arbitrary terms.

Input: Two terms s and t.

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

Algorithm: Start with the set of equations $\{s = t\}$.
(This is a singleton set: it has one element.)

As long as some equation in the set has one of the six forms listed on the next slide, perform the corresponding action.

Unification: Solving Term Equations

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

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

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:

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

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

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

Term Equations as Substitutions

The process of solving term equations always halts.

When it halts without reporting 'failure', the term equation system is left in a normal form: On the left-hand sides we have variables only, and they are all different. Moreover, these variables occur nowhere in the right-hand sides.

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 most general unifier for the input terms s and t.

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

Resolvents

Recall how we defined resolvents for propositional logic:

- Two literals are complementary if one is L and the other is $\neg L$.
- The resolvent of two clauses containing complementary literals L, $\neg L$ is their union omitting L and $\neg L$.

For predicate logic we have:

- Two literals L and $\neg L'$ are complementary if $\{L, L'\}$ is unifiable.
- Let C_1 and C_2 be clauses, renamed apart. Let θ be an mgu of complementary literals $\{L, \neg L'\}$ with L a literal in C_1 and $\neg L'$ a literal in C_2. Then the resolvent of C_1 and C_2 is the union $\theta(C_1 \setminus \{L\}) \cup \theta(C_2 \setminus \{\neg L'\})$.

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

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

- Colin is a large white fish living in deep water

$$W(colin) \land D(colin)$$

- 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

- $\forall x\ (S(x) \Rightarrow \exists y\ (T(y) \land E(x, y)))$ $\quad \{\neg S(x), T(f(x))\}$
- $\{\neg S(x), E(x, f(x))\}$

All large white fish are sharks

- $\forall x\ (W(x) \Rightarrow S(x))$ $\quad \{\neg W(x), S(x)\}$

Colin is a large white fish living in deep water

- $W(colin) \land D(colin)$ $\quad \{W(colin)\}$
- $\{D(colin)\}$

Any tadpole eaten by a deep water fish is miserable

- $\forall z\ ((T(z) \land \exists y\ (D(y) \land E(y, z))) \Rightarrow M(z))$
- $\{\neg T(z), \neg D(y), \neg E(y, z), M(z)\}$

Negation of: Some tadpole is miserable

- $\neg \exists z\ (T(z) \land M(z))$ $\quad \{\neg T(z), \neg M(z)\}$

A Refutation

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

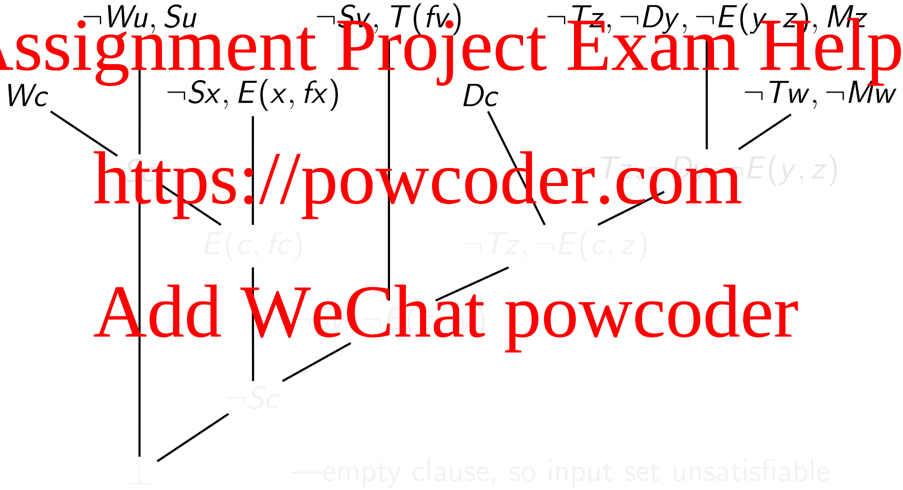
To save space, we leave out braces and some parentheses, for example, we write $\neg Wu, Su$ for clause $\{\neg W(u), S(u)\}$.

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

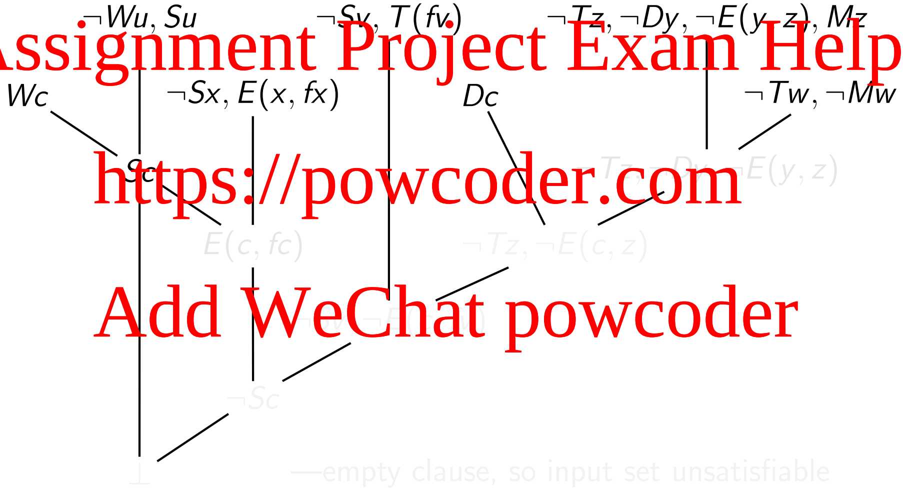
$$Wc \qquad \neg Sx, E(x, fx) \qquad Dc \qquad \neg Tw, \neg Mw$$

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

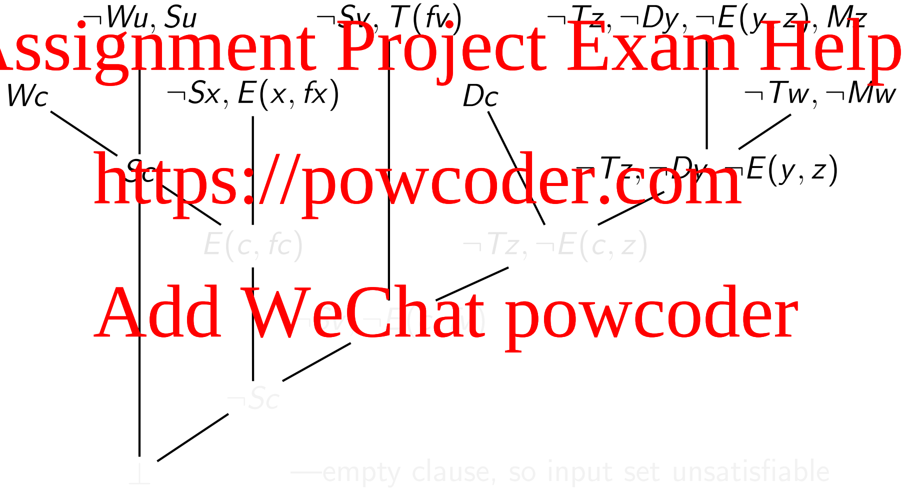
A Refutation for the Tadpole Example



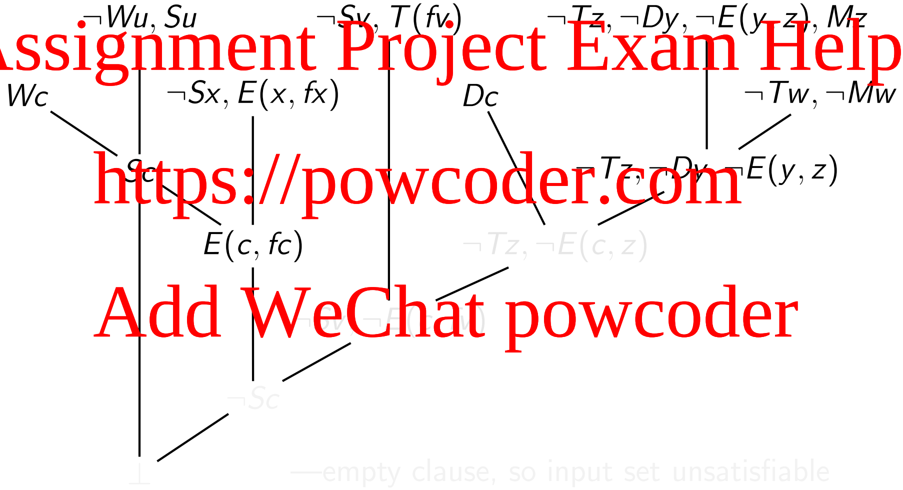
A Refutation for the Tadpole Example



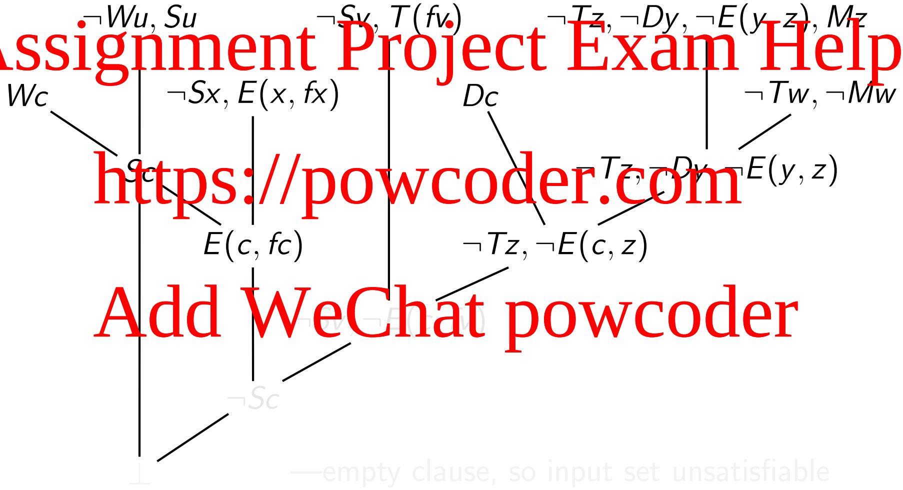
A Refutation for the Tadpole Example



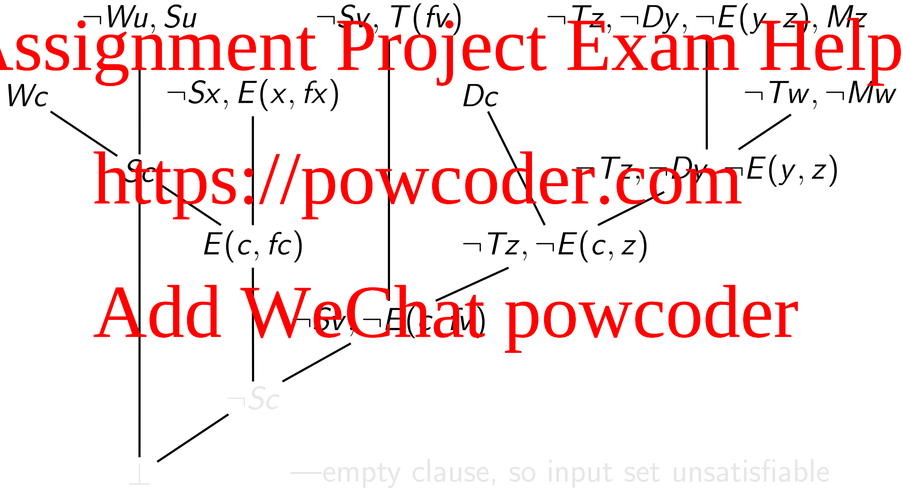
A Refutation for the Tadpole Example



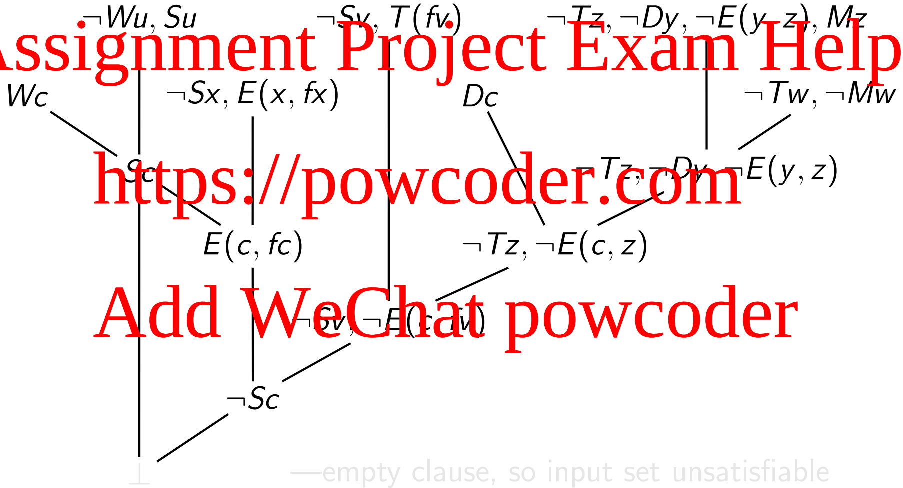
A Refutation for the Tadpole Example



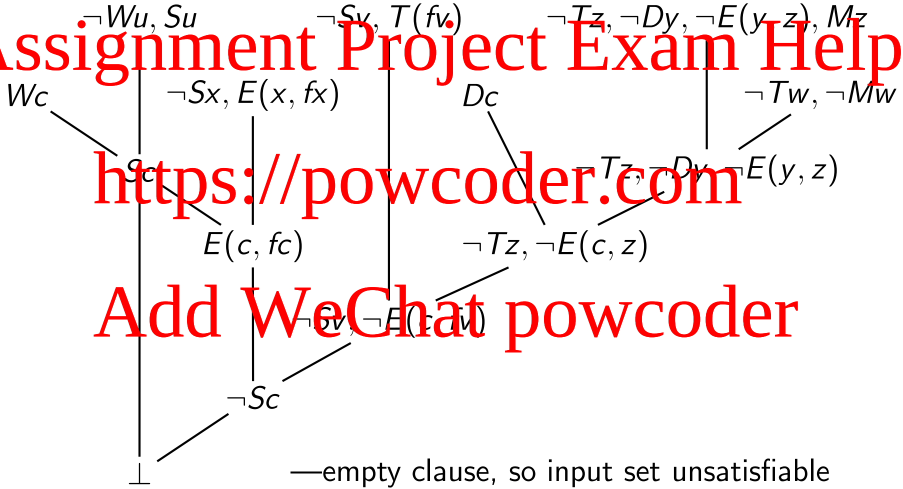
A Refutation for the Tadpole Example



A Refutation for the Tadpole Example



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 $\exists x\,(G(x) \wedge P(x))$
- Therefore some Greeks are wise $\exists x\,(G(x) \wedge W(x))$

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 $\theta(C)$.

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

$$\Big|$$

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

$$\Big|$$

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

Factoring is sometimes crucial:

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

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.

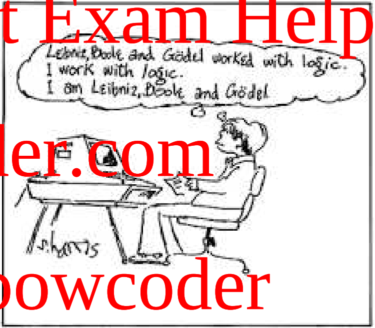
Hence we really should rename all variables in a clause every time we use the clause, using fresh variable names.

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

Leibniz's Dream

The only way to rectify our reasonings is to make them as tangible as those of the Mathematicians, so that we can find our error at a glance, and when there are disputes among persons, we can simply say: Let us calculate, without further ado, to see who is right.

G. W. Leibniz, 1685



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 $C_1, C_2 \in \mathcal{C}$

Does this process always terminate?

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 complete.

It gives us a proof procedure for unsatisfiability (and validity).

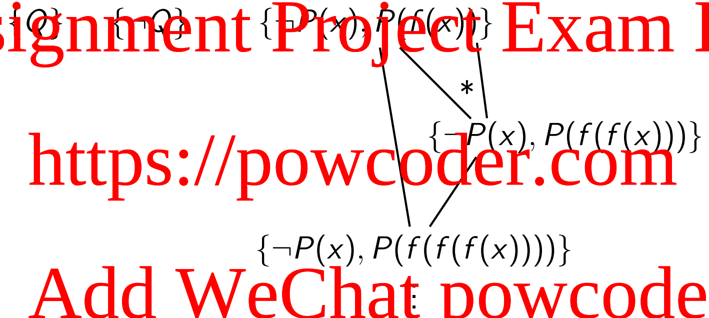
Note, however, that resolution does not give a decision procedure for unsatisfiability (or validity) of first-order predicate logic formulas.

Indeed, it can be shown that there are no such proof procedures.

We say that validity and unsatisfiability are semi-decidable) properties.

Proof Search

Resolution only works if we apply a sensible search strategy:



Moreover, search can be expensive.

* Here we used two copies of the same clause for resolution—the second using fresh variables, say $\{\neg P(y), P(f(y))\}$. When the resolvent is later used, it too is first renamed.

Proof Search Strategies

There is a chapter on resolution proofs on the LMS under “Readings Online”.

It covers different search strategies, and it also has a longer explanation of why resolution is sound and complete. (For the latter it uses concepts of Herbrand interpretation and semantic tree.)

These parts are not examinable.

Jacques Herbrand, 1908–1931

Horn Clauses and Prolog

A Horn clause is a clause with at most one positive literal.

All seven clauses used in the tadpole example were Horn.

A clause such as $\{\neg T(z), \neg D(y), \neg E(y, z), M(z)\}$ can also be thought of as

$$(T(z) \land D(y) \land E(y, z)) \Rightarrow M(z)$$

In the logic programming language Prolog, the clause is written


```
miserable(Z) :- tadpole(Z),
                deepwaterfish(Y),
                eats(Y,Z).
```

Horn Clauses and Search

For logic programs, the restriction to Horn clauses simplifies the search problem, but it does not remove it.

For propositional Horn clauses, satisfiability can be decided in linear time. (For arbitrary propositional CNF it is NP-complete.)

Sidebar: Unification and Type Inference

Another important use of unification is in type checking/inference. Here variables are type variables and the function symbols are type constructors. For example, think of the list type constructor as ‘List’, so we write ‘`list(x)`’ instead of the Haskell type ‘`[x]`’, and let us write ‘`fun(x,y)`’ instead of ‘`x -> y`’:

```
map  :: fun(fun(x,y), fun(list(x), list(y)))
null :: fun(list(z), bool)
```

If we use the expression `map null "abc"` then the type checker effectively sets up these equations:

```
fun(x,y) = fun(list(z), bool) -- for map null
list(x)  = list(char)         -- for (map null) "abc"
```

Sidebar: Unification and Type Inference

The unification algorithm then produces these equations:

```
x = list(z)
y = bool
x = char
```


and hence

```
list(z) = char
```


Unification failure shows that `map null "abc"` is not well-typed.

Next Up

Next we'll look at mathematical proof techniques.

Next week: We establish some basic discrete mathematics concepts and results: sets, relations and functions.